INTEGRITY VERIFICATION FOR SCADA DEVICES USING BLOOM FILTERS AND DEEP PACKET INSPECTION

THESIS

Michael W. Doroski, Captain, USAF

AFIT-ENG-14-M-25

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-14-M-25

INTEGRITY VERIFICATION FOR SCADA DEVICES USING BLOOM FILTERS AND DEEP PACKET INSPECTION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Michael W. Doroski, B.S.C.E.

Captain, USAF

March 2014

AFIT-ENG-14-M-25

INTEGRITY VERIFICATION FOR SCADA DEVICES

USING BLOOM FILTERS AND DEEP PACKET INSPECTION

Michael W. Doroski, B.S.C.E.
Captain, USAF

Approved:

//signed//	10 Mar 2014
Barry E. Mullins, PhD (Chairman)	Date
//signed//	10 Mar 2014
LTC Robert J. McTasney, PhD (Member)	Date
//signed//	10 Mar 2014
Maj Jonathan W. Butts, PhD (Member)	Date

AFIT-ENG-14-M-25

Abstract

In the past, supervisory control and data acquisition (SCADA) networks were made secure through undocumented, proprietary protocols and isolation from other networks. Today, modern information technology (IT) solutions have provided a means to enhance remote access through use of the Internet. Unfortunately, opening SCADA networks to the Internet has provided routes of attack. Cyber attacks on these networks are becoming more common and can inflict considerable damage to critical infrastructure systems. Furthermore, devices on these networks can be infected with malware that causes them to falsify their responses to operators, concealing alternate operation or hiding alarm conditions. Considering their applications, securing these networks translates to improved physical security in the real world.

Since modern IT solutions are impractical to deploy in the resource constrained SCADA networks, other solutions must be researched. This research evaluates an integrity verification system implemented on a Xilinx ML507 development board called the SCADA IntEgrity VErification (SIEVE) system. The design incorporates Bloom filters and SCADA-specific intrusion detection techniques to speed identification of invalid commands and current sensing to investigate whether or not a device correctly carried out a given command.

Results show that the SIEVE system is able to inspect and correctly identify 100% of network traffic at a 200 command per second frequency. Correct identification of valid MODBUS/TCP traffic begins to fail at 350 commands per second, introducing false positives. Tests of the Bloom filters show that they reduce the time necessary to process and log invalid MODBUS/TCP commands by 4.5% to 2328.06% depending on the number of operations performed by the command.

Acknowledgments

Many thanks to my advisor, Dr. Mullins, for keeping me focused and pointing me in the right direction while conducting this research.

I’d also like to thank Pranav Patel for helping me figure out the correct way design a digital system and that crossing clock domains isn’t as simple as it seems.

Michael W. Doroski

Table of Contents

Page

Page

Page

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
ADC	Analog to Digital Converter
ADU	Application Data Unit
CUT	Component Under Test
DAC	Digital to Analog Converter
FPGA	Field Programmable Gate Array
HMI	Human-Machine Interface
ICS	Industrial Control System
IDS	Intrusion Detection System
IED	Intelligent Electronic Device
MTU	Master Terminal Unit
PDU	Protocol Data Unit
PLC	Programmable Logic Controller
RDIVS	Remote Device Integrity Verification System
RTU	Remote Terminal Unit
SCADA	supervisory control and data acquisition
SIEVE	SCADA IntEgrity VErification
SUT	System Under Test
VHDL	VHSIC Hardware Description Language

INTEGRITY VERIFICATION FOR SCADA DEVICES USING BLOOM FILTERS AND DEEP PACKET INSPECTION

I. Introduction

1.1 Motivation

Behind the scenes, computer networks run many of society's critical infrastructure systems. In the past, these networks were secured through isolation and proprietary protocols. Now, modern information technology (IT) networking solutions and use of the Internet are being employed in order to increase efficiency through better remote access and ease of implementation. Unfortunately, this new paradigm reduces the efficacy of the prior security boons. Even worse, network defense solutions for modern IT networks do not translate effectively to these Industrial Control Systems (ICSs).

ICS networks are comprised of distributed devices that correspond with a central control point. Both the central control point and distributed devices are vulnerable to attack when exposed to the open internet. Defending these networks has become ever more important as cyber attacks result in physical effects in the real world [MW13, SFS11].

Aside from attacks originating outside of ICS networks, operators must now consider attacks from *inside* the network. Discovered in June 2010, the STUXNET malware not only infected computers in the these networks, but also the distributed devices [FMC11]. It overwrote portions of the devices' code and caused them to falsify their reports to the network operators. This would cause the operators to believe that the network is in a safe state when in fact it could actually be unstable or dangerous.

Since these networks are used in critical applications, where invalid operation causes real world effects, hardening their cyber security translates into improved physical security.

1.2 Overview and Goals

At present, most research focuses on detection of invalid network traffic going to the distributed devices. Little has been done to detect devices compromised by malware that causes them to falsify their responses [FMC11]. This research implements a system to provide realtime remote device security through integrity verification. This is accomplished through use of intrusion detection techniques and direct signal measurement. The proposed system is implemented using an embedded system approach, combining both hardware and software. Since the program structure of a supervisory control and data acquisition (SCADA) device is relatively static (not changing often), a hardware implementation approach for this system makes sense.

Two experimental goals are established for this research. The first is to determine if the proposed system can in fact correctly process all packet types in realtime, which is defined as within five milliseconds. The second goal is to determine if the use of a probabilistic data structure that is efficient in both space occupied and lookup speed provides any benefit for processing packets. The hypotheses for these goals is that the proposed system will be able to correctly process all packet types in under 5 ms and that properly configured Bloom filters will improve processing time for invalid commands. Five metrics are gathered in order to assess these goals: command processing time, response processing time, total processing time, count of packets inspected, and count of packets logged.

1.3 Thesis Layout

This chapter presents the motivation for the research and outlines the research goals. Chapter 2 covers the background of the topic, presenting overviews in SCADA Network security, Intrusion Detection technologies, and Bloom filters. Chapter 3 explains the operation of theproposed system, its design criteria, and its detailed design. Chapter 4 presents the test methodology for the evaluation of the proposed system. Discussion and analysis of the results of the experiments are presented in Chapter 5. Overall conclusions and future work are discussed in Chapter 6.

II. Background

This chapter provides an overview of SCADA networks, intrusion detection, Bloom filters, and research related to these three topics. Topics relating to SCADA networks such as their composition, vulnerabilities, and attacks against them are covered in Section 2.1. Section 2.2 covers the basics of intrusion detection techniques as well as the advantages of hardware based implementations of these services. Section 2.3 introduces the concepts central to the operation of Bloom filters and their applications. Lastly, Section 2.4 presents research related to the covered topics.

2.1 SCADA Networks

2.1.1 Overview.

SCADA networks provide a human operator with the ability to control and monitor geographically distributed sensors and actuators. The benefits of this sort of control are realized when the remote sensors or actuators are not only widely distributed but difficult to reach as well [Boy10]. These networks are operated around the world by both private industry and national governments and are used to control manufacturing processes and critical infrastructure [SFS11]. Examples of the critical infrastructure industries that employ SCADA networks are water and wastewater, oil and gas, electricity, and transportation.

A SCADA network is comprised of many different electronic or mechanical devices. The two primary components of such a network are the Master Terminal Unit (MTU) and the geographically-dispersed intelligent control devices. The remote controllers are subsequently connected to multiple sensors or other controllers at their respective field sites. Today, three types of intelligent control devices predominate: Remote Terminal Units (RTUs), Programmable Logic Controllers (PLCs), and Intelligent Electronic

Devices (IEDs). The controllers differ in their control circuit complexity and amount of devices that they can control [SFS11, Boy10]. One of the ways in which these controllers interact with other devices is through current loops. The most predominant implementation of this control method is the 4 to 20 milliamp loop [Sol]. Communication is achieved by one party driving the current in the loop to different values while the other party reads the changing value and responds accordingly. For the 4-20 mA loop, 4 mA represents the smallest value while 20 mA represents the largest.

The MTU serves as the master SCADA server, distributing commands to the remote devices and receiving data gathered from their sensors or controls. As shown in Figure 2.1, it resides in a centralized location along with the Human-Machine Interface (HMI), Data Historian, and I/O Server. The HMI is the means by which an operator can send commands to the SCADA network as well as view data collected by the remote devices. The Data Historian simply stores and timestamps data collected from the sensors and is usually a standalone computer [SFS11]. Lastly the I/O Server serves as the interface between the MTU and the remote devices and is generally a router or network switch.

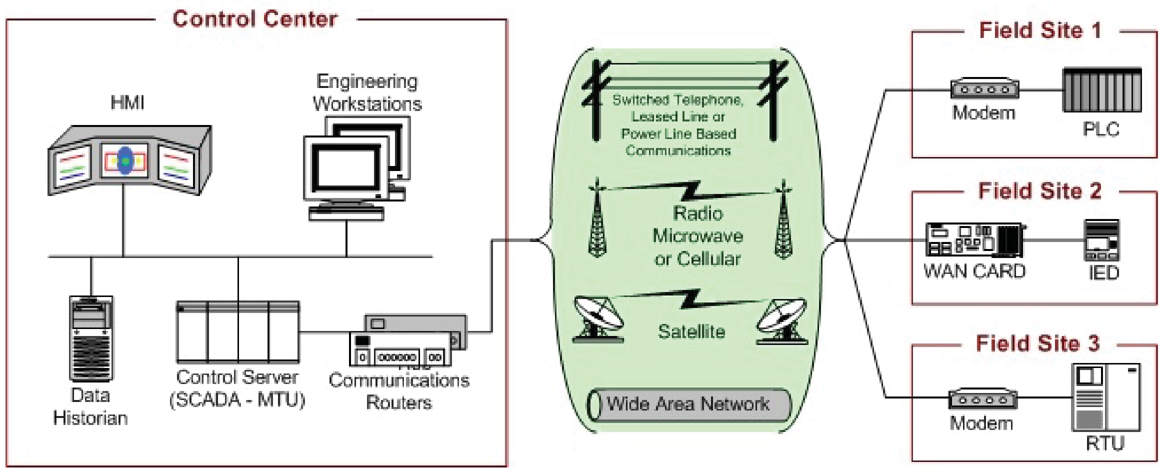


Figure 2.1: Example SCADA Network [SFS11]

Besides the network composition, SCADA networks differ from traditional IT networks in several key ways. First and foremost, SCADA networks are hard realtime systems [SFS11]; that is, systems in which an operation completed after its deadline is useless [ZJS11]. This implies that SCADA networks are *highly* intolerant to latency or disruption. In a traditional IT network, delays in the millisecond range are tolerable and do not affect normal operations whereas in a SCADA network, a 5 ms delay could mean the difference between a valve opening in time to release pressure from a water pipe or a burst water main. Reliability requirements for a SCADA network are also much higher than in traditional IT. So much higher in fact that even rebooting a computer would be intolerable. Lastly, much of the software, communication, and overall operation of SCADA networks are proprietary, causing traditional diagnosis and remedy of issues to be either ineffective or very difficult [AONR12, SFS11]. Consequently, due to how SCADA networks are employed, any disruption in functionality could lead not only to human quality of life issues but also safety risks as well.

2.1.2 Protocols.

SCADA Networks use a wide variety of proprietary protocols in order to communicate. Examples include Ethernet/IP, DeviceNet, ControlNet, PROFIBUS, MODBUS/TCP, DNP3, and Foundation Fieldbus [ILW06]. Of these SCADA protocols, MODBUS/TCP and DNP3 are very widely implemented and documented [Dut07, EBPS09].

This work will focuses on analysis and detection of attacks implemented using the MODBUS/TCP protocol.

2.1.2.1 MODBUS.

MODBUS is a serial communications protocol introduced in 1979 by Modicon [Dut07]. Originally implemented to allow serial communication between intelligent control devices, it has now been established as one of the standard protocols for industrial control networks. The original MODBUS specification implements the lowest two layers of the OSI model [Mod12] – the physical layer and the link layer. Application layer communications are enabled through the use of the MODBUS Application Protocol [Dut07]. Communication using this protocol follows a master–slave relationship, that is only the master can initiate communication, the slave merely responds. In SCADA networks, the MTU is the master and the remote devices are slaves, allowing the use of a shared communication medium, depicted as the Wide Area Networks in Figure 2.1.

The master–slave relationship of communication using the MODBUS Application Protocol allows for an ordered exchange of data frames between the client and server [Mod12]. As shown in Figure 2.2, a MODBUS Application Protocol frame, also called an Application Data Unit (ADU), consists of a Protocol Data Unit (PDU), device specific addressing information, and error checking fields [Mod12]. The ADU provides device specific addressing and error correction while the PDU contains the function code and related data. Overall, the ADU can represent either a command to a specific MODBUS server or a response to a client from the same.

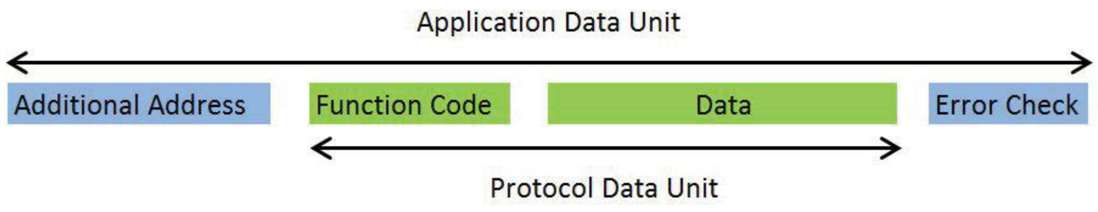


Figure 2.2: General MODBUS Frame [Mod12]

On a serial line such as RS232, the additional fields of the ADU add to the total length of the frame. Due to limitations of the medium, the maximum size of an ADU is 256 bytes. If the additional address field is one byte long and the error check field is two bytes, then the maximum size the PDU can have is 253 bytes [Mod12].

The PDU represents either a command from a client that a server should carry out or a response from a server to a client. It is encoded as a single byte function code and various related data. The MODBUS protocol defines three different types of PDU:

- MODBUS Request PDU
- MODBUS Response PDU
- MODBUS Exception Response PDU

The first type of PDU, the MODBUS Request PDU, is the only PDU created by clients. The other two PDUs, the MODBUS Response PDU and MODBUS Exception Response PDU, are created by MODBUS servers and are responses to a Request PDU. For each PDU type, the function code and related data have different meanings. The function code in the Request PDU represents a command for the device to execute, in a Response PDU it was the command that was executed, and in an Exception Response PDU it is the command the device attempted to execute but failed to do so correctly [Mod12]. In a properly functioning SCADA network, the MODBUS Exception Response PDU should never be present because invalid commands should never be issued to a device.

As a one byte field, 256 different function codes are available. However, the MODBUS Application Protocol reserves the upper 128 codes as exception responses. This is because if an Exception Response PDU is generated, its function code field is set to the original function code plus 0x80 [Mod12]. An example is if a function code 0x01 resulted in an exception, the Exception Response PDU's function code would be set to 0x81. In a normal response, the Response PDU's function code is set to the same as received in the Request PDU. An example is the function code of a Response PDU for a Request PDU with function code 0x01 is 0x01 [Mod12]. Additionally, the MODBUS

Application Protocol predefines 20 different function codes [Mod12]. All other function codes are available for user defined commands.

The MODBUS Application Protocol also defines four types of data constructs which can be operated on with function codes. Two of them, the coil and the discrete input, are single bit constructs and the second two, the holding register and input register, are 16 bit registers. Discrete inputs and input registers can only be read from, while coils and holding registers can be written to as well as read. Conceptually, the read-only constructs come from an external I/O system, an attached device perhaps, while coils and holding registers are sourced by commands from a client, providing control information for the controller itself. About half of MODBUS' predefined function codes utilize these constructs [Mod12]. The predefined function codes refer to these data constructs using a two byte address field where the addresses of the four constructs do not overlap. Thus, a coil referred to at address 0x0000 is different from the analog input at address 0x0000. Consequently, a device is theoretically able to control 2^{16} of each data construct.

2.1.2.2 MODBUS/TCP.

MODBUS/TCP is an embedding of the MODBUS serial-line protocol into TCP/IP. This allows operators to control devices over a modern packet switched network rather than direct links only. Figure 2.3 depicts both the MODBUS Application Protocol stack and the MODBUS/TCP stack. As can be seen, for standard MODBUS, frames produced in the application layer are transmitted directly over RS232 or RS485. For MODBUS/TCP, some translation is done then the frames are wrapped in standard TCP/IP packets and transmitted over Ethernet [Mod06].

Figure 2.4 depicts the basic MODBUS/TCP ADU. Instead of device-specific addressing information as in the MODBUS serial line frame, a new field is added called the MODBUS Application Protocol (MBAP) Header. The MBAP header is seven bytes long and contains a transaction identifier, protocol identifier, number of following bytes,

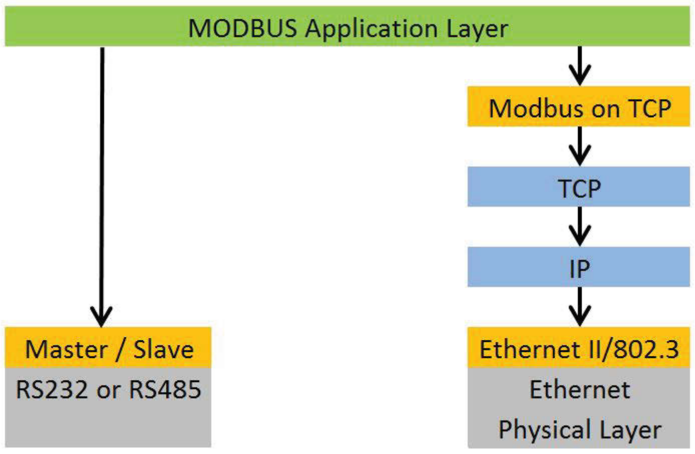


Figure 2.3: MODBUS Communication Stack [Mod12]

and a unit identifier. Table 2.1 contains a listing of the fields in the MBAP header, their byte locations, and a description of their contents. The PDU field of the MODBUS/TCP ADU is the same as the MODBUS serial-line protocol. Due to this, the MODBUS/TCP PDU also has a maximum length of 253 bytes. Therefore, including the MBAP Header, TCP header, IP header, and Ethernet frame header, the maximum size of a MODBUS TCP/IP packet is 314 bytes, that is 253 bytes (PDU) plus 7 bytes (MBAP header) plus 20 bytes (TCP header) plus 20 bytes (IP header) plus 14 bytes (Ethernet frame header).

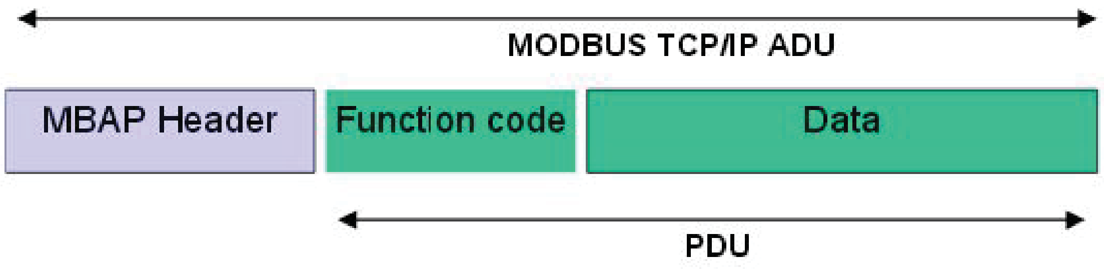


Figure 2.4: MODBUS/TCP ADU [Mod06]

Table 2.1: MBAP Header

Field	Byte Numbers	Description
Transaction Identifier	0-1	Numerical identifier for the current transaction
Protocol	2-3	Protocol Descriptor, set to 0x00 for MODBUS/TCP
Length	4-5	Total number of following bytes
Unit Identifier	6	Unique device identifier, set to 0xFF if not used

According to the MODBUS/TCP specification, all MODBUS/TCP packets are sent to TCP port 502 [Mod06]. In this way, MODBUS/TCP traffic can be identified by its destination port.

2.1.2.3 DNP3.

DNP3 was developed by Westronic Incorporated between 1992 and 1994 [dnp12]. Westronic Incorporated intended DNP3 to be the first “truly open” protocol standard for use in the utility industry and tried to include as many of the best features of other industrial protocols along with newly created features. Two specific requirements were addressed in the design of the protocol: the need for scalability and emphasis on reliability. The first requirement resulted in the use of as few layers of the OSI model as possible with a consequent reduction of necessary bandwidth (due to fewer headers). The second requirement resulted in a frame with two cyclic redundancy check bytes for every 16 data bytes, allowing for detailed error detection at the devices, or outstations [dnp12].

Device connections in DNP3, unlike MODBUS, are all referred to as points. Points can fall into one of five categories with an even larger selection of types called variations. The five categories are binary inputs, analog inputs, counter inputs, control outputs, and analog outputs. For instance, the analog input category has six variations which are listed below in Table 2.2. Similar to the MODBUS addressing scheme, each category of point is individually indexed using integers [dnp12].

Table 2.2: Analog Input Variations

Variation Number	Description
1	32-bit integer value with flag
2	16-bit integer value with flag
3	32-bit integer value
4	16-bit integer value
5	32-bit floating-point value with flag
6	64-bit floating-point value with flag

As stated above, DNP3 does not implement all layers of the OSI model in order to reduce bandwidth consumption. It consists of a layer 2 protocol that provides segment encapsulation into data link frames, data link frame decoding into transport segments, error detection, and source and destination addressing, a layer 4 like transport function that allows for packet fragmentation, and a layer 7 that defines functions and data types [dnp12]. Thus, DNP3 can handle larger or more complex commands than MODBUS.

Communication between master and outstation in DNP3 is different from MODBUS in that outstations can generate spontaneous responses to the master. The spontaneous responses are usually generated in order to inform the master of some event that has occurred and should take notice of [dnp12]. This does not quite break the master–slave communication relationship as spontaneous responses are more like informing that master that some communication needs to happen.

Also like MODBUS, DNP3 frames can be embedded in TCP/IP packets to be sent over packet-switched networks.

2.1.3 Vulnerabilities.

When SCADA was originally implemented, network security was not a great concern. Isolation from external networks and undocumented proprietary protocols provided adequate protection from attack [HS05, ZJS11]. Improvements over the past twenty years in network technology and the global connections provided by the Internet allow SCADA networks to be run much more efficiently through remote administration from any Internet-connected computer and at lower cost, however this is at the risk of being open to the world [ILW06]. However due to the proprietary nature, hard realtime requirements, and availability requirements of SCADA networks, traditional IT network defense is extremely difficult, if not impossible, to employ [CWY08, ZJS11].

Stouffer classifies SCADA vulnerabilities into three categories: policy and procedure, platform, and network [SFS11]. Policy and procedure vulnerabilities result from improperly designed or executed procedures. An example is having no security procedures for the network. Platform vulnerabilities are more hardware or software oriented. These vulnerabilities are inherent to the devices or software and can be very difficult to protect against. Examples of this type of vulnerability include software buffer overflows allowing remote code execution, insecure remote access to the devices, or physical damage. Zhu describes this category of vulnerabilities as “extremely detrimental” [ZJS11]. Mitigation strategies include implementing proper security controls such as software patching and physical access control. The last category of vulnerability is network. Network vulnerabilities occur due to network flaws, improper configurations, or incorrect administration of the network itself or other connected networks. An example of a network vulnerability is transmission of passwords in clear text. This class of vulnerability is protected against through careful network design and good network administration.

Ten models the vulnerability of SCADA networks through analytical means [TLM08]. His model is represented using an extensible markup language (XML) file with simulations accomplished using custom Visual Basic.Net code and MATLAB. The simulations take into account system vulnerability, scenario vulnerability, and access point vulnerability. His results suggest that taking basic security precautions such as implementing a stronger login policy will lower the vulnerability of a given access point. The IEEE suggests security should be balanced with operations and cover access and use control, data integrity and confidentiality, restricting data flow, responding timely to events, and ensuring network availability [IEE08].

2.1.4 Attacks.

SCADA networks have been increasingly subject to cyber attack. A congressional report from March 2013 on electric grid vulnerability reveals that the industry is under constant cyber attack [MW13]. The attackers also may not necessarily be individuals but nation states as well [NWD+12].

According to Zhu, there are three different types of targets in a SCADA network: hardware, software, and communication stack [ZJS11]. Attacks on hardware include obtaining remote or physical access to the system and changing the set points and changing the data that the operator of the network sees. This first attack could result in causing the hardware to fail or an alarm to not go off while the second could result in concealing alarm conditions or generating false alarms. A software attack takes advantage of vulnerabilities in the software being run either by the remote devices or by the central command center. Communication stack attacks are manifested more as denial of service type attacks such as attempting to delay data or rerouting commands or requests to unintended recipients. In fact, Yang demonstrated that SCADA networks are vulnerable to man-in-the-middle attacks via ARP spoofing [YML+12].

In the past ten years, several significant attacks on SCADA networks have been realized. The most severe may be the STUXNET malware discovered in 2010 [AONR12, ZJS11, NWD$^+$12]. STUXNET spread across systems using four different Windows 0-day vulnerabilities and searched for specific Siemens PLC software installed on the system. If found, it modified the PLC code in order to sabotage the system. Specifically, it caused the PLCs to execute rogue commands while replaying valid, recorded data to the operators. This caused the PLCs to damage the devices they were controlling while the operators believed everything was normal. The only way to detect the invalid operation of the PLCs was to directly observe the operation of the controlled devices. The danger this sort of malware attack presents is that it is very difficult to detect and can cause severe damage to the systems or to human life.

Attacks on SCADA networks need not be targeted. In 2003, the SQLSlammer Worm infected over 75,000 computers. One system that became infected was the Davis-Besse nuclear power plant in Ohio. Slammer managed to crash the SCADA network's display and monitoring system for almost five hours despite the existence of a firewall [NWD$^+$12]. This caused performance issues as the system was not operating as expected. Consequently, had an unsafe situation developed at the plant, the problem may have been hidden from operators, possibly causing it to spiral out of control.

2.2 Intrusion Detection

2.2.1 Overview.

An Intrusion Detection System (IDS) is either a device or software that examines traffic on a network in an attempt to identify anomalous events [VM08]. It is important to note that intrusion detection does not cause the packets to drop; instead traffic that violates the network rules are flagged and an alarm is raised. In traditional IT, IDSs use deep packet inspection to look for either anomalous traffic or known attack signatures. According to Zhu, these techniques both rely on primary evidence such as semantic definitions, access policies, or abstraction of known illegal patterns [ZS10]. Consequently, intrusion detection in a SCADA environment can be difficult to employ due to hard-to-define illegal patterns or abnormal behavior.

A further problem that exists in employing intrusion detection in a SCADA environment is the lack of computing resources. SCADA devices are commonly designed to operate within the bare minimum of hardware and may not have the additional resources required to operate an intrusion detection program [SFS11].

2.2.2 Techniques.

2.2.2.1 Non-SCADA Specific.

The two primary techniques for intrusion detection used by both traditional IT and SCADA networks are signature-based detection and anomaly-based detection. Signature-based detection focuses on detecting malformed packets and known attack payloads through packet inspection. Figure 2.5 shows an example of the type of packet a signature-based Intrusion Detection System would detect [VM08]. The packet on the left is a correctly created and well formed MODBUS/TCP command packet whereas the packet on the right has had its data length and data fields tampered with. In this case, a PLC would read the malformed packet possibly causing a buffer overflow that could be exploited.

Anomaly-based detection is also extendable to SCADA networks. This detection technique is looking for network traffic patterns that do not match the norm with the key requirement being consistent and predictable traffic patterns, which SCADA networks exhibit. In this case, an anomaly-based detector is trained on normal traffic data and then deployed into an operational environment. Once deployed it searches for traffic that is not similar to what it has been trained on [LMV12]. Many researchers are actively investigating this technique in relation to SCADA networks [LMV12, VM08, PK12].

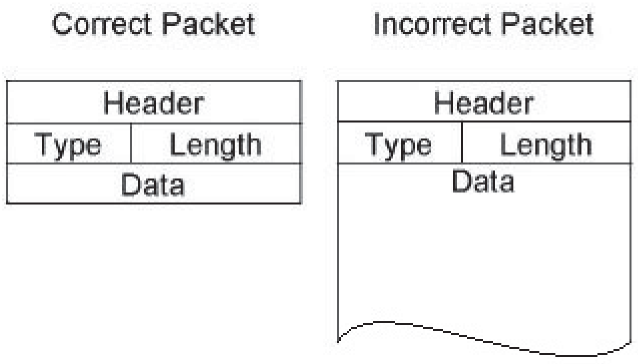


Figure 2.5: Correct and Incorrect MODBUS Packet [VM08]

2.2.2.2 SCADA Specific.

One of the primary purposes of a SCADA network is to monitor the state of a large scale process. Assuming legitimate operators would never intentionally put a system into a critical state, that is a state that could harm people or damage equipment, predicting the effect a command could have on a system would be an accurate method of determining if an intrusion took place [CCG+11]. This form of detection seeks to model the state of the system and how it evolves as new MODBUS commands arrive, sending an alert as the system moves into a critical state. Carcano implemented a prototype IDS that implements this method and tested it in a system composed of three PLCs each connected to a sensor and a controller [CCG+11]. The prototype demonstrated a 0.35% false positive rate and a 0.05% false negative rate. In a separate performance test where the prototype was connected to 16 PLCs each connected to "at least" 100 different analog or digital devices, he found calculation time grew linearly with rules and number of components per rule. In the extreme case of 2000 different rules, the prototype performed each state update in 57.386 ms.

Another SCADA-specific technique is specification-based detection. Cheung proposed the creation of rules using models to generate expected traffic on a system

[CDF+07]. This is a form of whitelisting, and ensures that all traffic in a network is actually well defined and valid. Morris used the MODBUS specification guides [Mod06, Mod12] to define a set of 50 different signature-based rules [MJVD13] that can be integrated easily with popular software-based intrusion detection systems like Snort.

2.2.3 Hardware Implementations versus Software Implementations.

The platform upon which an IDS is implemented is generally not as important as the optimization and speed of the IDS software. However, as stated above, SCADA networks are constrained in terms of processing speed, memory, and network speed; they simply do not have the resources to run additional software. Additionally, SCADA networks have hard realtime requirements; completing a request on time is of paramount importance [SFS11]; thus the IDS must be able to inspect traffic in millisecond time frames. Hardware-based IDSs developed on Field Programmable Gate Array (FPGA)s offer these types of performance capabilities. The benefits that an FPGA offers is parallelism and the ability to develop specialized hardware to support the algorithm.

Signature matching is an algorithm that is highly parallelizable. This is possible because a packet need not be compared to only one other item at a time. Dharmapurikar accomplishes this through the use of parallel Bloom filters [DKSL04]. He implemented a prototype system for matching one string against 10,038 other strings and achieved a throughput of 2.46 Gbps. The parallel Bloom filters only used 14% of the FPGA fabric of a Xilinx XCV2000E as well. Sourdis took the concept further and implemented custom pipelined comparators and encoders [SP05]. Each digital structure in his design translates directly to a specific rule for the popular IDS software Snort. After implementing a total of 210 structures (or rules), his system achieved a throughput of 8.064 Gbps and used 71% of the FPGA area in a Xilinx Virtex2-6000. He speculates that through more optimization another implementation of this method could fit more structures into the FPGA and achieve higher throughputs.

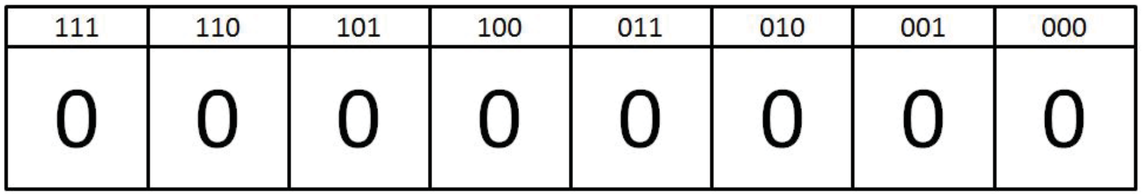
2.3 Bloom filters

2.3.1 Overview.

Bloom filters are a space-efficient probabilistic data structure proposed by Bloom in 1970 [Blo70]. Their intended use is to test whether a message is a member of a particular set of messages. In this case, messages refer to data structures represented by a string of bits. Their probabilistic nature manifests in that false positives are a possible outcome of membership queries. This drawback is offset by the space saving of the structure itself as well as the time-efficient query operation. Unlike false positives, false negatives are *not* possible, thus if a negative is returned on a query the object is definitely not in the set.

In traditional hash tables a message is hashed using some algorithm to get an index in the table. Next, the table entry at the index is checked to see if it is empty. If it is, the original message is stored. If it is not empty, the hashed message is hashed again using the same algorithm in order to get a new address. This process is repeated until an empty address is found for the message [Blo70]. A query operation is similar in that the message to be queried is hashed to get an index then the message at that index is compared to the hashed message. If they match, the message is in the set. If it does not match, the index must be hashed and the next location checked. This operation repeats until either the message or an empty location is found. Both operations entail several executions of the hashing algorithm and, depending on how full the hash table is, could take many clock cycles to complete.

Instead of a table, a Bloom filter consists of a bit array m bits wide all initialized to '0' and k different hash functions that produce hashes ranging from 0 to $m-1$. Both m and k must be whole numbers and greater than or equal to 1. A newly created Bloom filter with m=8 is depicted in Figure 2.6. An insertion is accomplished using the k different hash functions to get k different addresses in the bit array and setting those entries to '1', regardless of their current value. Figure 2.7 depicts an example insertion of a message into an eight bit wide Bloom filter with k equal to two. A lookup is performed by first hashing a message using the same k hash functions as in the insertion operation. If each entry corresponding to the resulting k addresses is '1', the message may be in the set. Figure 2.8 shows a lookup of two messages, Message_1 and Message_2. Message_1 had previously been inserted into the filter in Figure 2.8 while Message_2 has not been inserted at all. This lookup example also demonstrates the impossibility of false negatives. A single '0' bit from any address results in a negative lookup. Due to hash collisions, a positive lookup result does not necessarily indicate that the message is in the set. An example of this can be seen in Figure 2.9 – this is the false positive problem [Blo70].



111	110	101	100	011	010	001	000
0	0	0	0	0	0	0	0

Figure 2.6: Uninitialized Bloom Filter

The false positive rate p is

$$p = \left(1 - e^{\frac{n*k}{m}}\right)^k \tag{2.1}$$

where m is the number of bits in the filter, n is the the number of messages in the set, and k is the number of hash functions. Generally, assuming a fixed m, the false positive rate increases as messages are inserted into the filter and decreases as the number of hash functions k or array width m increases. Dharmapurikar, in [DKSL04], used

$$p = \left(\frac{1}{2}\right)^k \tag{2.2}$$

as a starting point for minimizing p. By substituting Equation 2.2 for p in Equation 2.1, k can be solved for in terms of m and n, resulting in

$$k = \left(\frac{m}{n}\right) * ln(2) \tag{2.3}$$

When k is chosen in this fashion, the false positive rate becomes the original substition for p, Equation 2.2 [DKSL04]. This also allows k to be determined as a function of the ratio of bit array width m and number of messages n.

Unlike hash tables, deletions are not possible in standard Bloom filters. This is due to presence in the Bloom filter being represented as a single bit. In the case where an address relates to one item only, setting the stored bit to '0' is not a problem. In the case where the address is shared, setting the stored bit to '0' affects other items in the filter, introducing false negatives. Different Bloom filter implementations have been developed [FCAB00, MLVD12] to circumvent this problem and are described below.

Another important consideration when implementing a Bloom filter is the hashing algorithm. Often the hash algorithm calculations are time consuming, taking more than one clock cycle to compute. Along with computation speed, a small probability of collisions is desirable [PK12]. A class of hashing algorithms that fits these requirements is H_3 [CW79]. Algorithms that belong to H_3 create hashes by performing a logical AND on a random bit vector for each bit in a message. The resultant vectors are subsequently exclusive ORed together to obtain a single hash. As shown in Figure 2.10, each bit of a message is logically ANDed with a corresponding random vector. The results of these AND operations are subsequently exclusive ORed together to determine a final hash. The three-byte random vectors are chosen to show the final result is different from the inputs. Ramakrishna determined this class of hashing algorithms is ideal for hardware-specific implementation due to ease of translation into physical structure and inherent parallelism [RFB97]. Figure 2.11 depicts such an implementation where the x_i values are input bits from a message and $q_{i,j}$ values are bits from the random vectors. For Bloom filters, the random bit vectors can be $log_2(m)$ bit long numbers, where m is the number of bits in the filter, such that the final hash represents the address of a bit [DKSL04].

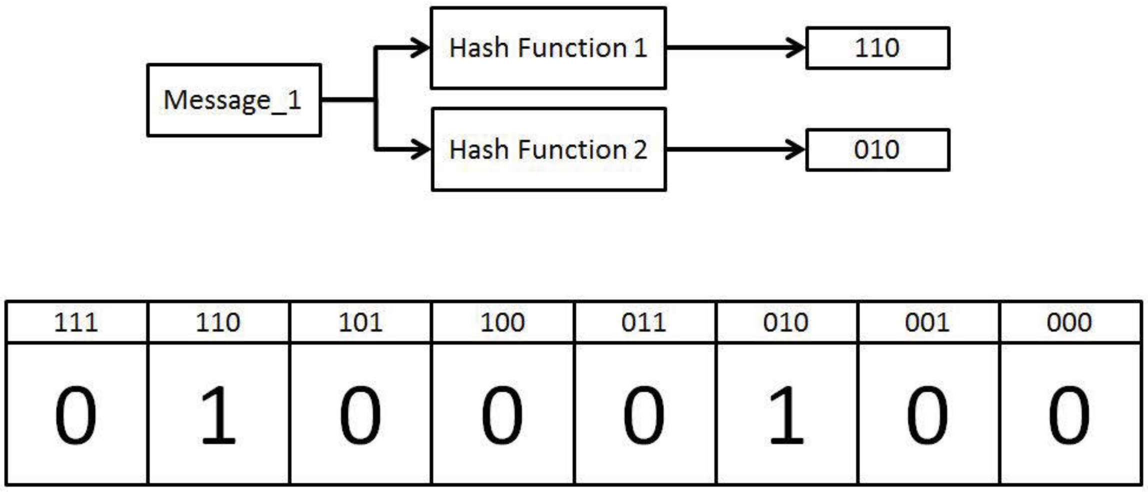


Figure 2.7: Bloom Filter Insertion Example

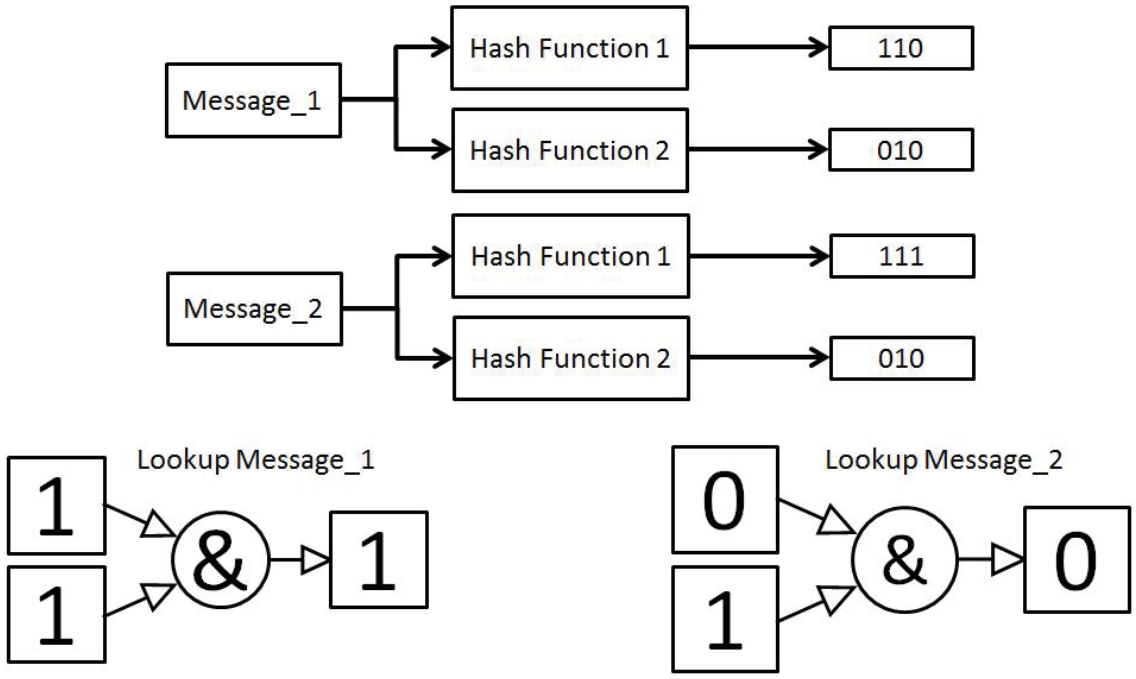


Figure 2.8: Bloom Filter Lookup Example

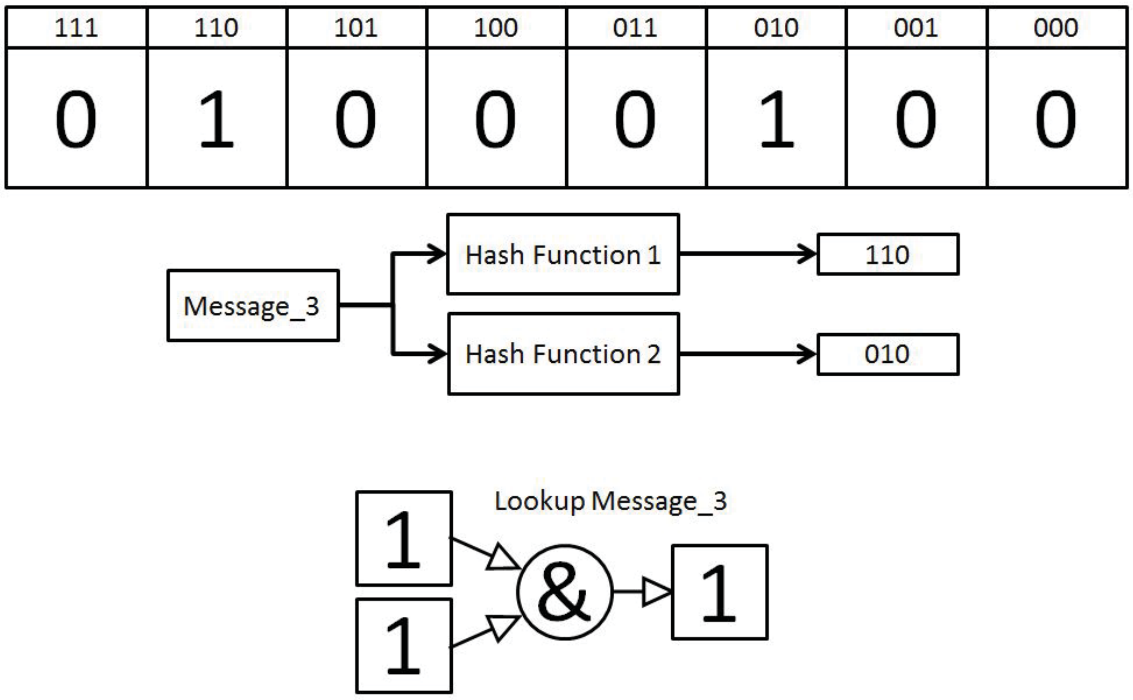


Figure 2.9: Bloom Filter False Positive Example

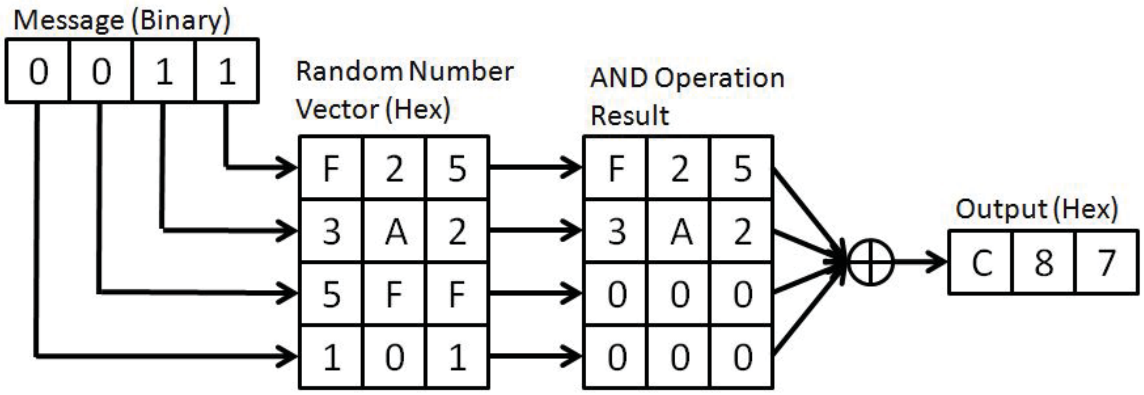


Figure 2.10: H_3 Algorithm Example

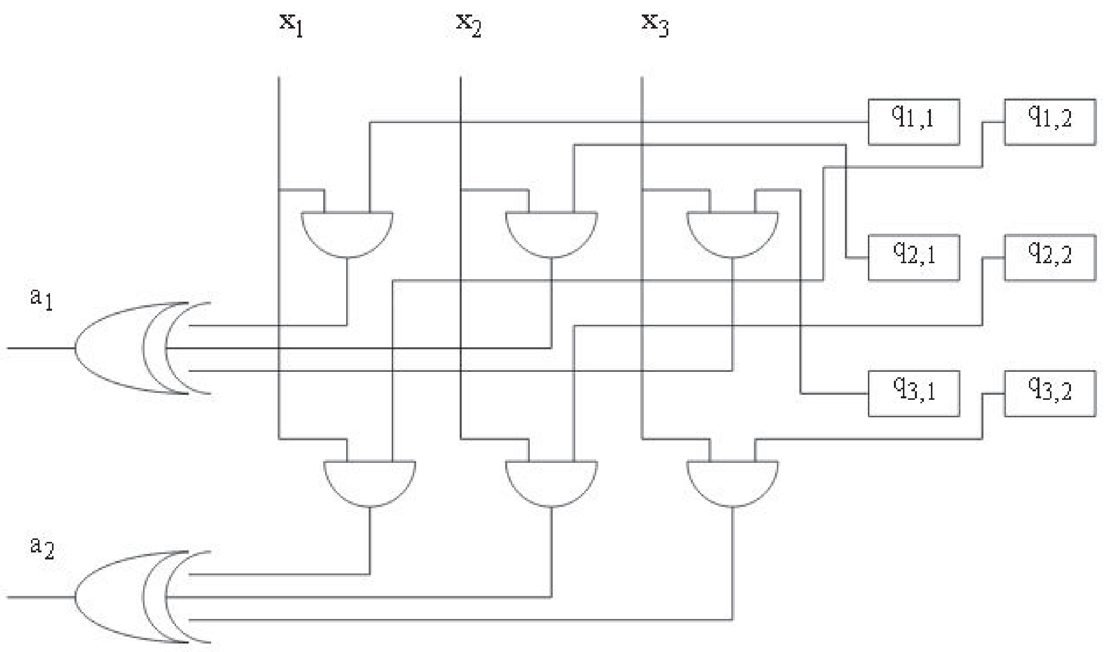


Figure 2.11: Circuit implementation of an H_3 Algorithm [RFB97]

2.3.2 Types.

The counting Bloom filter attempts to overcome the problem of false negatives introduced by deleting entries in standard Bloom filters [FCAB00, MKR12, GLLY10]. In a counting Bloom filter, instead of the filter being a bit array, an array of counters is used. When an item is inserted, the counter is incremented and when one is deleted it is decremented. Querying a counting Bloom filter takes slightly more time as the results of a greater than zero comparison on the entries to be examined are used as input to the AND operation signifying presence in the filter instead of the contents of the entry alone. In situations where deletions are necessary, such as a filter whose set has members that can "age" out, this may be acceptable when slower access speed is not a factor [GLLY10]. Mukuntharaj characterized the performance impact in a hardware implementation that could achieve lookup throughputs in the 5 Gbps range [MKR12].

Other exotic Bloom filters have been created for various purposes. Moreira developed a Bloom filter he terms the concatenated Bloom filter that is resistant to saturation attacks [MLVD12]. A saturation attack is when enough items are inserted into the Bloom filter such that every entry is equal to ‘1’, thereby making it useless as a discriminator. The concatenated Bloom filter works by dividing the main filter into *d* subfilters each with size *m*/*d*. Elements are inserted into the filter one by one, each one going into a different subfilter. When the *d+1*th element is reached, it is inserted in to the first subfilter and the operations continue. By intelligently picking d, he demonstrated that the false positive rate can be bounded and never reach 1.

2.3.3 Applications.

Bloom filters have been used in a wide variety of applications. Recently they have been used for IP traffic analysis [LRBS12], discovery protocols [SMPMLVLS11], and intrusion detection [PK12, DKSL04, MKR12]. In each application they are slightly different. Lambruschini, Dharmapurikar, and Mukunthuraj use the filters as classifiers, indicating whether a tested packet is in one set or another. Parthasarathy uses them as sequence identifiers, indicating whether a sequence of commands is anomalous. In these applications the key is speed. Sanchez-Monedero exploits their other widely known benefit of compact size in order to ease network congestion from transmitting network endpoint information across a peer to peer network by using them as a compact storage medium that can be easily transmitted [SMPMLVLS11].

2.4 Related Research

2.4.1 Role-Based Authentication.

Role-based authentication has been implemented as one way of securing SCADA networks. Hieb created a security-hardened appliance in order to implement role-based authentication using a Gumstix microcontroller and Bloom filters [HSG13]. This device sits inline with the MTU and RTU, acting as a gate keeper. The microcontroller runs a form of Linux with a kernel that is difficult to attack due to compartmentalization and thorough debugging. With the device secure, a challenge-response authentication protocol is implemented. If an incoming command would cause a change in settings, the device responds with a challenge to the sender. The sender must be able to decipher the challenge and send a response with a hash-based message authentication code (HMAC). If the HMAC in the response matches the one the device calculated, the command is passed through to the device. Non-critical commands such as read coils do not need to be authenticated as they are not changing the system in any way. Bloom filters are used in this design as a way to look up commands. It employs two Bloom filters, the first to check if the command is valid and the second to determine if the command is critical, that is a command that would change the device's state. On the Gumstix device, a Bloom filter lookup took 18 μs. In testing, the device was able to stop unauthorized write attempts to the device but not read attempts. This is due to read commands being non-critical. The current implementation of the device takes 200 ms to generate the initial challenge, which could delay a command past its execution deadline.

2.4.2 Bloom Filter Anomaly Detection.

Work is also being done using Bloom filters and anomaly detection to prevent intrusions in smart grids [PK12]. Parthasarathy proposed an anomaly detection based IDS that takes into account system state. In his implementation, normal traffic is first analyzed using n-gram analysis. N-gram analysis analyzes sequences using a sliding window of size n. For his proposed system, he uses n=5 messages and splits the function codes and data into separate n-grams, yielding two types of sequences. These sequences are then inserted into two Bloom filters for fast lookup later. Upon receiving a new packet, it is inserted into the current test n-gram and looked up in the Bloom filters. Anomaly scores are assigned to both the function code n-gram and data n-gram which are subsequently added together to yield a composite anomaly score. His implementation showed a 2.4%

false negative rate for n=100 but 0% for n=200 and n=500. The system operates on the order of “a few ms” for varying sizes of data, demonstrating a constant lookup speed. This delay would likely be acceptable on a SCADA network.

2.4.3 Critical State Based Firewall.

Critical state-based detection is a new method of intrusion detection [CCG+11]. Fovino has taken the concept proposed by Carcano and implemented a critical state-based firewall [FCCM12]. The Fovino firewall consists of a rule analyzer and state controller. The state controller keeps track of the current state of the system by monitoring all traffic from the MTU to the remote device. Any change in state indicated by the device is reflected in the state controller’s virtual system image. The rule analyzer monitors the virtual system image searching for any critical states or evolution towards critical state using a calculation Fovino refers to as the state distance metric. In network performance tests, it adds less than two ms of additional latency which Fovino characterizes as negligible. The virtual system image can consume up to 400 MB of memory when 1000 PLCs are connected. The state distance metric suffers as more rules are added, taking over 50 ms to update. However, the 50 ms is not involved in blocking commands, only in raising alerts. In this sense, while slow, it can still be considered acceptable. Fovino has demonstrated that this type of detection is viable however requires a non-negligible amount of computing resources.

III. Device Overview

This chapter covers the design and operation of the SCADA IntEgrity VErification (SIEVE) system. Section 3.1 covers the basic operation of the system. Section 3.2 covers the criteria used to design the system. Lastly, Section 3.3 describes in depth the operation of custom-designed components.

3.1 System Overview

Integrity verification on SCADA networks is a different approach to intrusion detection. Intrusion detection on SCADA networks is a fast evolving field with many unique aspects [ZS10]. While many solutions have been proposed for detecting attacks in progress [ZS10], little has been proposed in order to detect already compromised devices. Furthermore, intrusion detection does not take into account the erstwhile operator who unknowingly sends a command that puts the system in an unsafe state. The SIEVE system is designed to fulfill these purposes: verifying the validity of incoming network traffic and the integrity of the PLC's response.

In order to verify the integrity the PLC, the SIEVE system must be efficient in its operations. This was achieved through integration of both software and hardware structures in an embedded design, implemented using a Xilinx ML507 development board and a custom designed current sensing circuit. The specific command processing method is specification-based, combining Cheung's Model Based Intrusion Detection [CDF+07] scheme with Morris' MODBUS/TCP Rules Based Intrusion Detection [MJVD13], and implemented with hardware elements similar to Sourdis' FPGA-based signature matching algorithm [SP05]. Rather than storing allowable commands in hardware arrays like Sourdis' FPGA, all valid commands are instead stored in a Bloom filter. Thus, when a command arrives it is dissected and looked up in a Bloom filter. If the Bloom filter lookup returns a negative result an alert is raised. If a positive result is returned, the packet is inspected again in software to confirm its validity due to possibility of false positives from the Bloom filter. This is necessary because any invalid commands sent to a PLC or other remote device could result in catastrophic failure.

Verifying the integrity of a response is more difficult than verifying the validity of a command. For responses, several steps are required. First, is to match the transaction ID and function code to the preceding command. Assuming the match is made, the response must be further processed to determine if any monitored lines were operated on. This can be difficult due to lack of information in the response packet itself. For instance, in read operations, the MODBUS command contains a base address and the number of subsequent reads to perform. An example is a read five coils command with base address 0x0000 means respond with the contents of coils at addresses 0x0000 - 0x0004. The response the PLC provides contains the specified data but not the base address nor how many coils were read, it is assumed that client that sent the command has that information. Consequently, if a valid command is received it must be stored but not logged as invalid in order to facilitate processing the response. Also, if the response proves to be invalid, the lack of information could preclude analysis as to why it was marked invalid. Thus, if a response is found to be invalid, the previous valid command is logged at the same time in order to provide context. Specific command and response compositions will be discussed below in Section 3.3.

Another aspect of verifying the validity of a response is ensuring that the PLC *actually* executed the command. The SIEVE system accomplishes this by monitoring specific I/O lines coming from or going to a PLC. The SIEVE system assumes these I/O lines are 4-20 mA current loops. The monitoring is performed by reading the actual value of a line and comparing it against what the PLC reports after a command is executed. The reading is performed using Analog to Digital Converters (ADCs) and current sense circuits. The operational specifics of the current sense circuits and ADCs are covered in Sections 3.3.2 and 3.3.3 respectively.

Figure 3.1 depicts the concept diagram for the SIEVE system. It monitors commands and responses to the PLC from the MTU and reads from the I/O lines going to or from other devices. A communication line separate from the line used for commands and responses is used a means of alerting the operators when suspicious activity is detected.

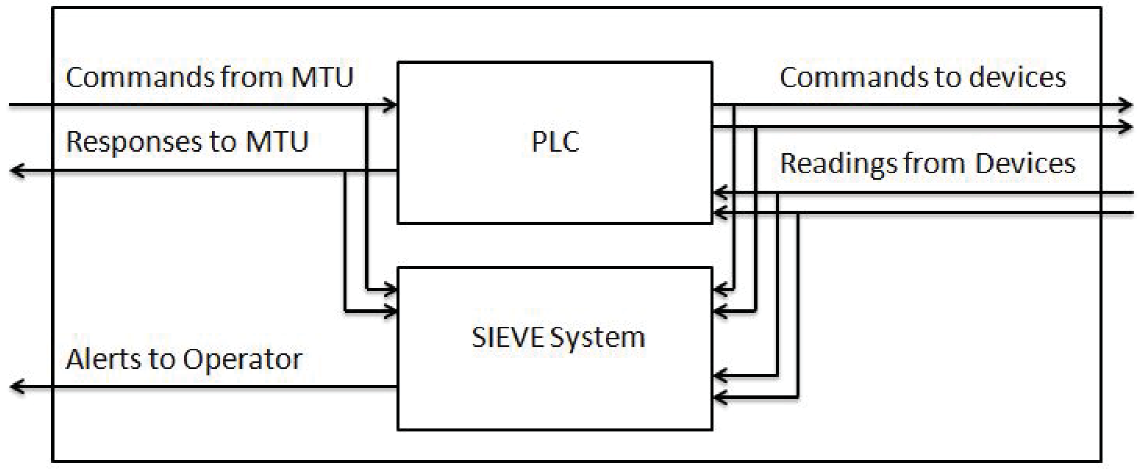


Figure 3.1: SIEVE System Concept

3.1.1 Algorithm.

The algorithm that the system operates on can be seen below in Figure 3.2. The basic flow is as follows:

1. Receive a packet going to or from the PLC.

2. Determine its direction in order to interpret it as a command or response.

3. Determine if it is MODBUS.

4. If it is not MODBUS, log the packet and report an error.

5. If it is a MODBUS command, process it through the Bloom filter.

   (a) If it is not in the Bloom filter, log the packet and report an error.

   (b) If it is in the Bloom filter, determine if a command is expected.

   (c) If a command is not expected, log the packet and report an error.

   (d) If a command is expected, process it through software.

   (e) If it is invalid, log the packet and report an error.

   (f) If it is valid, copy its information for processing the future response and expect a response for the next packet.

   (g) Wait for a new packet going to or from the PLC.

6. If the packet is a MODBUS response, determine if a response is expected

   (a) If a response is not expected, log the packet and report an error.

   (b) If a response is expected, determine if its header information matches the previous command's header information.

   (c) If the header information does not match the previous command's, log the previous command, log the response, and report an error.

   (d) If the header information does match the previous command's, determine if any lines that were operated on by the command are monitored.

   (e) If there are none, expect a command for the next packet and wait for the next packet to or from the PLC.

   (f) If one or more monitored addresses were operated on, query the ADCs.

   (g) If the ADC responses do not match the MODBUS response, log the previous command, log the response, and report an error.

(h) If the ADC responses match the MODBUS response, expect a command for the next packet and wait for the next packet to or from the PLC.

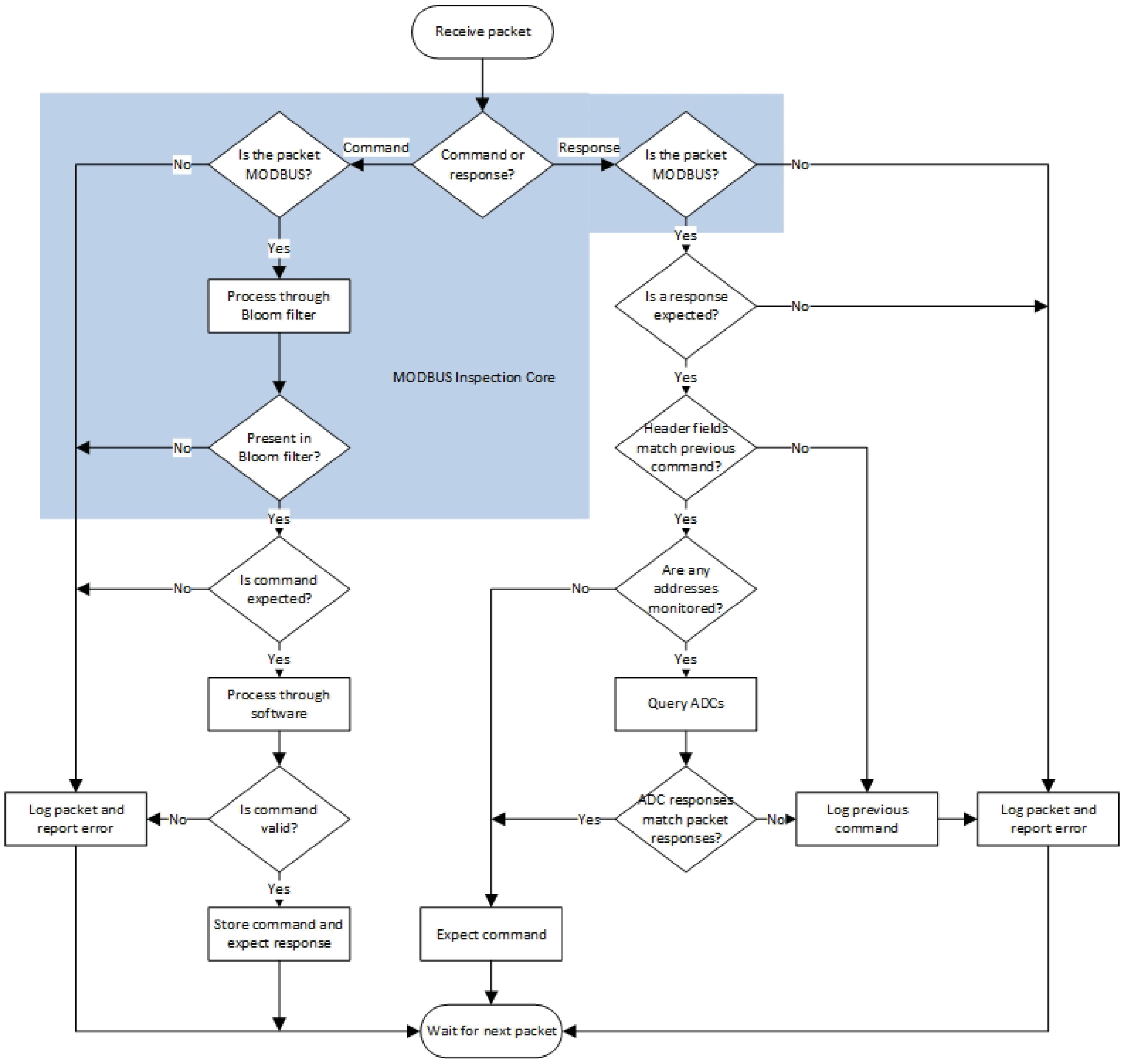


Figure 3.2: Packet Workflow

Three workflows come out of this algorithm: one for non-MODBUS/TCP packets, one for MODBUS/TCP commands, and one for MODBUS/TCP responses. The first workflow ends as soon as the packet is identified as not being MODBUS/TCP, meaning the offending packet is logged and a warning is sent. If the packet is identified as MODBUS/TCP it then flows down one of the other two workflows depending on its direction - that is, if its destination was the PLC then it is identified as a command and if its destination is not the PLC, then it is treated as a response. The destination MAC address is used for the direction decision because they are the first six bytes of the received ethernet frame.

For commands, the next step is further hardware processing in order to be sent through a Bloom filter. If all the operations in the command are present in the Bloom filter, then the command is passed back up to the software side of the system for further processing. If any one operation is not in the Bloom filter, then the packet is logged and a warning is sent. After being passed back to software, the command is processed in much the same way it was in the Bloom filter. If any one operation is not valid, then the command is considered invalid and the packet is logged with a warning sent to the operator. If the command proves to be valid, it is stored for use when processing the response.

If a MODBUS/TCP response is received, the first check is to determine if a response is expected. If it is not, the response is logged as a suspicious packet without logging the previous command. If a response was expected, the system proceeds to inspect it further. If its transaction ID or function code does not match those of the previous command, both command and response are logged. This is because it is unknown if the next response received will be valid for the current command – it in effect resets the state of the system to expect a command. If the response transaction ID and function code match the previous command, then its contents are processed to ensure the PLC correctly executed the command. This is done by loading the base address and iterating over all addresses operated on in the command. If an address matches a monitored line, the line's ADC is queried to get a reading as to the value of the line. If the reading does not match what was specified for the line, the response and previous command are logged. Otherwise, the response is valid and a command is expected next.

The command followed by a response structure of the algorithm derives from the MODBUS/TCP protocol itself [Mod12]. According to the protocol specification, new commands cannot be sent until a response, whether valid or exceptional, is received.

3.2 Design Considerations

The design of such a service involves at minimum the consideration of three criteria: speed, accuracy, and flexibility, which are defined more in depth below,

3.2.1 Speed.

Speed is one of the most critical factors in the design of the SIEVE system. Since SCADA networks are hard realtime systems, the analysis of any command, response, or packet going to or from the PLC before the execution deadline is critically important. The SIEVE system achieves this through both hardware and software methods. In fact, leveraging custom hardware greatly improves the speed of the system due to dedicated pipelines and minimization of command execution overhead.

3.2.2 Accuracy.

Accuracy is the other critical factor in the design of the SIEVE system. Identification of malicious or invalid commands are of paramount importance to protecting the operation of SCADA network. Consequently, every command, response, or non-MODBUS/TCP packet going to or from the PLC must be inspected and its validity verified. This means that under normal operating conditions, the SIEVE system must be able to correctly identify invalid activity 100% of the time. This is accomplished through the two stage hardware-software command verification and in-depth response analysis.

3.2.3 Flexibility.

The MODBUS Application Protocol allocates four 16-bit address ranges for the four basic data constructs: coils, discrete inputs, holding registers, and input registers. PLCs using MODBUS/TCP for communication must then use some or all of those ranges in order to correctly execute their programs. Given this, the SIEVE system must also be able to fully replicate those ranges in order to handle any command or response to or from the PLC it is monitoring. This is accomplished through use of off-chip memory as the Virtex 5 FPGA does not have enough onboard block RAM structures to store the necessary data.

3.3 System Design

3.3.1 MODBUS Inspection Core.

The MODBUS Inspection Core consists of nine hardware structures and is written in VHSIC Hardware Description Language (VHDL). Figure 3.3 depicts a simplified block diagram of their relations to each other. Data flows left to right except for the extracted function code which acts as a selector to the depicted multiplexer and demultiplexer. If an erroneous field is found while inspecting the packet, the entire core enters an error state, which alerts the CPU. This error state can only be reset by the CPU. Likewise, once a packet is completely inspected the finished state is only cleared through a reset by the CPU.

The Packet FIFO is the only non-custom designed hardware structure at the top level. The following sections describe in depth the operation of the other eight structures.

3.3.1.1 Packet Decode.

Figure 3.4 shows the state diagram of the Packet Decode block. In the Ready state, the Packet Decode block waits for a signal to begin decoding a packet stored in the Packet FIFO. This signal is provided to the core by the software running on the PowerPC CPU. After start goes high (logic ’1’) the block transitions to the Initialize state. The initialize state sets up the internal signals to begin operation, like resetting counters to 0. After one clock cycle, the block transitions to the Decode state. In this state, it reads the first 62 bytes of the packet from the Packet FIFO two bytes at a time. This takes 31 clock cycles

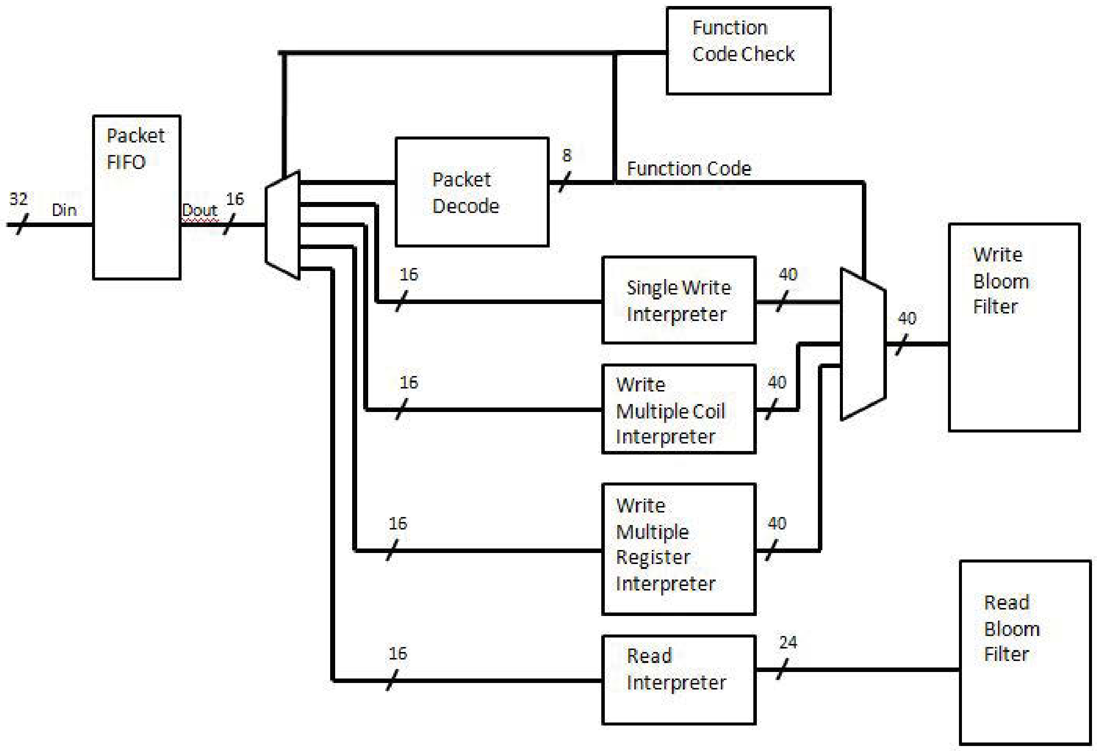


Figure 3.3: Simplified Block Diagram of the MODBUS Inspection Core

to complete. As the bytes are read, different fields are inspected to determine whether or not the packet is in fact MODBUS/TCP as well as if it is a command or a response. Table 3.1 contains the inspected fields and their byte numbers. Direction of the packet is determined by inspecting the first six bytes of the packet and comparing them to the PLC's MAC address, corresponding to the Ethernet frame's destination. If they are equal, it is a frame going to the PLC and should be interpreted as a command. Otherwise it is a frame coming from the PLC, and it is a response.

After determining direction, the next inspected field is the TCP destination port. As noted in Section 2.1.2.2, the TCP destination port should always be equal to integer 502. Thus, if the bytes are not equal to integer 502, the packet decode block transitions to its error state and raises an error. If the TCP destination port is equal to integer 502, inspection continues.

The next field inspected is the protocol ID in the MBAP header. As noted in Section 2.1.2.2, this should be set to integer 0. If the inspected bytes are not equal to 0, the block transitions to an error state and stops processing the packet. Otherwise, it continues processing.

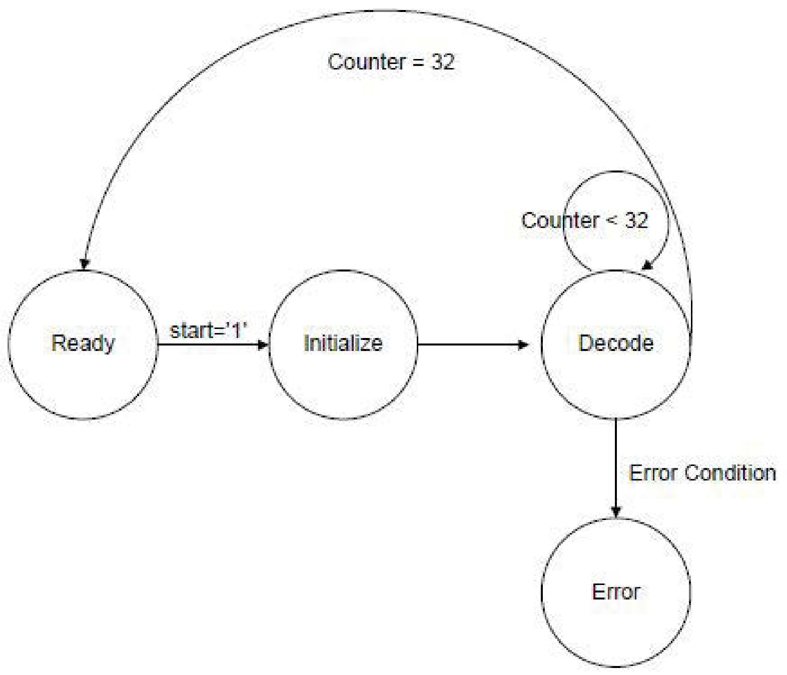


Figure 3.4: Packet Decode Block State Diagram

The last inspected field is the MBAP following length field. This field should be set to the packet length minus 60 as the last byte of this field is the 60th byte in the packet. If it is not, the block transitions to the error state. If this last field passes inspection, the function code is extracted to be used by the rest of the core. This occurs on clock cycle 31.

On the 32nd clock cycle, the finished signal goes high to signal the subsequent structures to continue inspecting the packet.

Table 3.1: MBAP Header

Byte Number	Field Description	Expected Value
0-5	Destination MAC Address	PLC MAC Address
36-37	TCP Destination Port	502
56-57	MBAP Protocol Identifier	0
58-59	MBAP Following Length	Packet Length - 60
61	Function Code	N/A

3.3.1.2 Function Code Check.

This is a simple hardware block that interprets the extracted function code. If the extracted function code is not in the block's lookup table, the block signals that the function code is invalid. This places the core in an error state.

3.3.1.3 Read Interpreter.

The read interpreter block handles all function codes that perform read operations. All read functions have the same command format: two bytes for the start address and two bytes for the number of read operations to perform. Table 3.2 contains the supported functions and their maximum number of operations.

Figure 3.5 shows the state diagram of the Read Interpreter Block. The Read Interpreter functions in much the same way as the Packet Decode Block. Initially it begins in the Ready state. After packet decode has been completed, the finished signal from the packet decode block is routed to the start signal of the read interpreter through a demultiplexer. When this signal goes high, the block transitions to the Initialize state.

Table 3.2: Read Function Codes

Function Code	Description	Maximum Operations
0x01	Read Coils	2000
0x02	Read Discrete Inputs	2000
0x03	Read Holding Registers	125
0x04	Read Input Registers	125

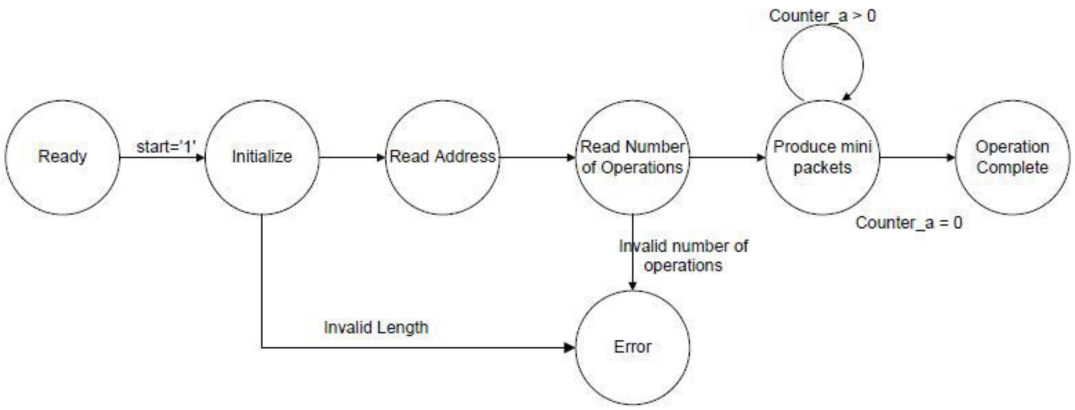


Figure 3.5: Read Interpreter Block State Diagram

In the Initialize state the internal signals and counters of the block are set to their initial value. The packet length is also checked to confirm that it is the correct length of packet for a read operation: 66 bytes. If it is not the correct length, the core transitions to the Error state and processing of the packet halts. If the length is correct, the interpreter transitions to the Read Address state. The next two bytes of the packet are the base address of the operation and must be stored. After storing the base address, the interpreter transitions to the Read Number of Operations state. The final two bytes of the packet are the number of consecutive reads to perform. This field must be at least one but less than or equal to the maximum number of operations shown in Table 3.2. If the number of operations is not in bounds, the core transitions to the Error state. Otherwise, the core transitions to the Produce Mini Packets state.

A mini packet is an interpretation of one operation that a MODBUS/TCP command performs. Created for the SIEVE system, they are used to translate commands into a series of operations that can be individually hashed and looked up in the Bloom filters. This allows for flexibility in the composition of commands going to the PLC without having to hash and store every single permutation of operations. Read mini packets are 24 bits (three bytes wide) and consist of the function code concatenated by the address to operate on. The interpreter calculates these by incrementing the base address until the number of operations has been reached. For example, a Read Coils command with base address 0x0005 and number of operations equal to three results in three mini packets: 0x010005, 0x010006, and 0x010007. After all mini packets have been produced, the interpreter transitions to an Operation Complete state.

If at any time during the operation of the interpreter the read Bloom filter indicates that a mini packet is not in the filter, the core transitions to the Error state.

3.3.1.4 Single Write Interpreter.

The Single Write Interpreter handles both single write commands: Write Single Coil and Write Single Holding Register. Their function codes are 0x05 and 0x06 respectively. Despite the coil being a single bit and the holding register being a two byte register, both commands have the same format. The first two bytes are the base address and the following two bytes are the data to be written. For coils, the only two acceptable values are 0xFF00 or 0x0000, representing logic ‘1’ and ‘0’ respectively.

Figure 3.6 shows the state diagram of the Single Write Interpreter. The interpreter begins operation in the Ready state. After the Packet Decode block has finished its operations, the finished signal is routed to the start input of the Single Write Interpreter.

When this signal goes high, the interpreter transitions to the Initialize state. In this state internal signals are reset to their initial values. The length of the packet is also checked to ensure it is the correct length of a single write packet: 66 bytes. If the length is not correct, the core transitions to the Error state.

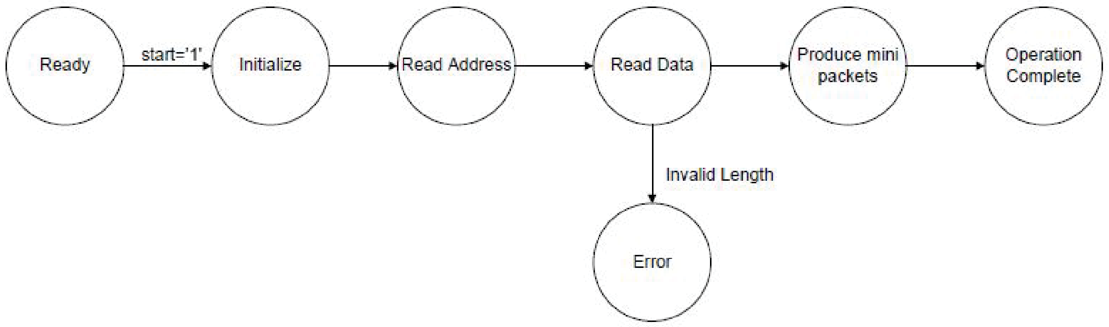


Figure 3.6: Write Single Interpreter State Diagram

If the length is correct, the core transitions to the Read Address state. The two bytes read from the packet in this state are the address which is to be written. After they are stored in a register, the core transitions to the Read Data state. The final two bytes of the packet are the data to be written to either the coil or register. These bytes are stored in a register and the core transitions to the Produce Mini Packets state.

Much like the read interpreter, the single write interpreter also translates the MODBUS/TCP packet into a more compact form for the Bloom filter. For writes, this is a 40 bit (five bytes wide) mini packet. It consists of the function code concatenated with the address field, which is in turn concatenated with the data field. After this mini packet is output to the write Bloom filter, the core transitions to an Operation Complete state.

If at any time during the operation of this core the Bloom filter establishes that a command is not in the filter, the core transitions to the Error state.

3.3.1.5 Write Multiple Coil Interpreter.

The Write Multiple Coil Interpreter is the first of two interpreters that handle a single function code. This is due to the command's unique structure. The Write Multiple Coils command is function code 0x0F and requires the most processing of all the implemented commands. The command is structured as follows: two-byte base address, two-byte number of operations, one-byte following byte count, and lastly byte count number of bytes of data. Unlike in the Write Single Command, data for writing multiple coils is packed bitwise into a byte rather than two bytes representing a single bit. Consequently, for each operation the correct bit needs to be extracted in order to create the correct mini packets. The maximum number of operations in the Write Multiple Coils command is 1968.

Figure 3.7 shows the state diagram of the Write Multiple Coil Interpreter. The Write Multiple Coil Interpreter begins operation in the Ready state. When the Packet Decode block finishes its operation, if the extracted function code was 0x0F, the finished signal is routed to the start signal of the write multiple coil interpreter. When this signal goes high, the interpreter transitions to the Initialize state.

In the Initialize state, the interpreter resets all internal signals and counters to their initial values. After one clock cycle it transitions to the Read Address state. The next two bytes read from the packet are the base address of the operation. After these are stored for later use, the interpreter transitions to the Read Number of Operations state. After the base address, the next two bytes of the packet are the number of operations to perform. If these bytes are less than one or more than 1968, the core transitions to the Error state. Otherwise, it transitions to the Read Byte Count state. The byte count is equal to the number of operations divided by eight or the number of operations divided by eight plus one. The latter condition occurs when the number of operations is not evenly divisible by eight. If the byte count is less than one or greater than 246, or the length of the packet

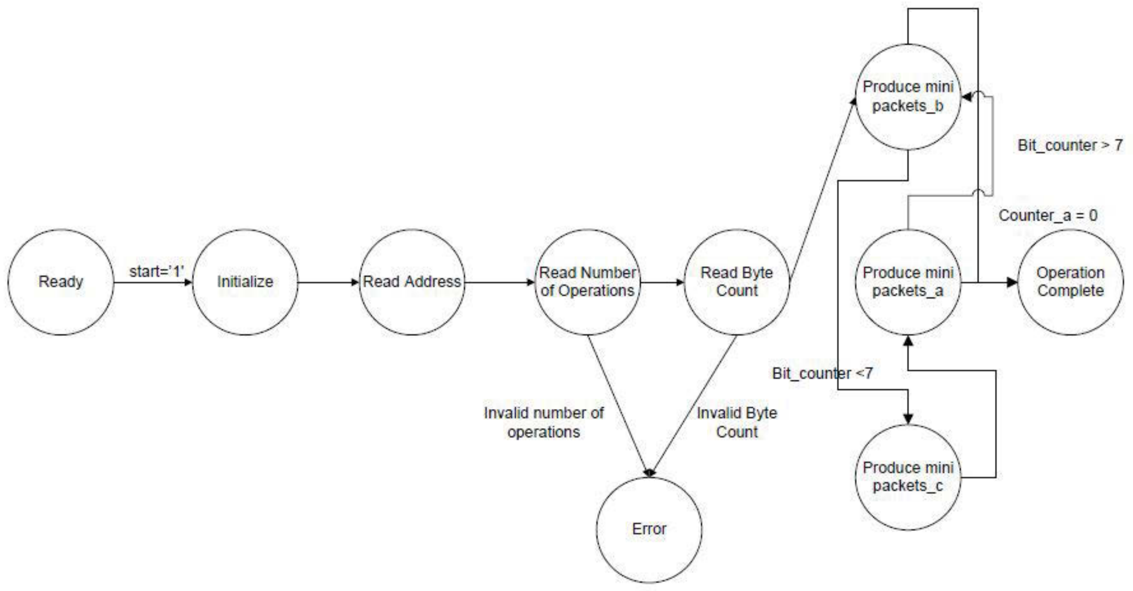


Figure 3.7: Write Multiple Coils Interpreter State Diagram

minus 67 does not equal the byte count, then the core transitions to the Error state. Otherwise, it transitions to the first Produce Mini Packets state.

After reading the byte count, the core transitions to the Produce Mini Packets_B state. This is due to reading the packet two bytes at a time and when the byte count is read, the data byte is also read. For the three Produce Mini Packets states, two counters are important. The first is the bit counter which keeps track of which bit is being inspected. When in Produce Mini Packets_B, the bit counter is less than eight and it iterates through the lower byte that was read. After it has interpreted eight bits, it transitions to Produce Mini Packets_C which triggers the next read from the Packet FIFO. After reading the next two bytes, the core transitions to Produce Mini Packets_A. In Produce Mini Packets_A, the bit counter is greater than eight and the upper byte of the two bytes from the packet is interpreted. When that byte is exhausted, the state transitions to Produce Mini Packets_B and the process begins again. The second counter that is being updated through all the previous transitions maintains how many mini packets have been produced. The Write Multiple Coils core produces the same write mini packets as the single write interpreter, five bytes wide. When the current bit of data being inspected is ‘1’, the mini packet that is created has its data field set to 0xFF00. If the current bit inspected is ‘0’, the data field is set to 0x0000. Once the total number of mini packets needed to be produced are created, the core transitions to the Operation Complete state.

If at any time during the operation of this core the Bloom filter establishes that a command is not in the filter, the core transitions to the Error state.

3.3.1.6 Write Multiple Register Interpreter.

The Write Multiple Register Interpreter is the second of the two interpreters to interpret one command only. The Write Multiple Holding Registers command is structured similarly to the Write Multiple Coils command. Its function code is 0x10 and is structured as follows: two-byte base address, two-byte number of operations, one-byte byte count, and then byte count data bytes. Processing this command is easier than Write Multiple Coils because the data to be written to a holding register is encoded as two bytes rather than one bit.

Figure 3.8 shows the state diagram of the Write Multiple Register Interpreter. The Write Multiple Register Interpreter begins operation in the Ready state. When the Packet Decode block finishes its operation, if the extracted function code was 0x10, the finished signal is routed to the start signal of the Write Multiple Register Interpreter. When this signal goes high, the interpreter transitions to the Initialize state.

In the Initialize state, the Write Multiple Register Interpreter resets its internal signals and counters before beginning processing. After one clock cycle it transitions to the Read Address state. The next two bytes of the packet are the base address for the operation. After they are stored for later use, the interpreter transitions to the Read Number of Operations state. Following the base address, the next two bytes of the packet are the

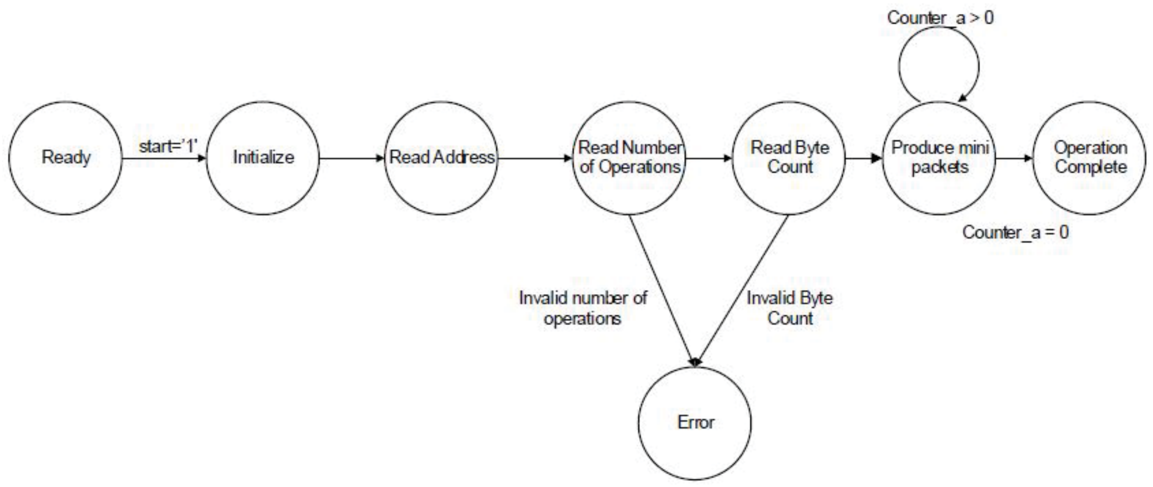


Figure 3.8: Write Multiple Holding Registers Interpreter State Diagram

number of operations to perform. This field must not be less than one or greater than 123. If either condition is true, the core transitions to an error state. Otherwise, the number of operations is stored and the core transitions to the Read Byte Count state. Since each set of data for each register is two bytes, the byte count should be equal to two times the number of operations. If this is not true or the byte count is not equal to the size of the packet minus 67 the core transitions to the Error state. Otherwise, it transitions to the Produce Mini Packets state.

In the Produce Mini Packets state the interpreter continues to read from the Packet FIFO and outputs a new mini packet at the end of each clock cycle. Once the total number of operations mini packets are produced, the core transitions to the Operation Complete state.

If at any time during the operation of this core the Bloom filter establishes that a command is not in the filter, the core transitions to the Error state.

3.3.1.7 Bloom Filters.

The Bloom filters are the crux of the command inspection in the hardware stage of operation. Rather than taking multiple clock cycles to look up data ranges or if a command is valid for an address, the Bloom filter can perform the lookup and verify whether a command is invalid or probably valid.

Figure 3.9 depicts a simplified block diagram of the implemented Bloom filters. At their core they consist of a hash matrix, five Block RAMs, and one large AND gate. The hash matrix is set up in the same way as in Figure 2.11 where each bit of the input string either zeros out a random vector or allows it to proceed on to the XOR gate. It is written to be completely independent of the clock and uses ten different H_3 hash algorithms. The ten H_3 algorithms produce addresses used to look entries up in the Block RAMs. The Block RAMs are Xilinx cores that have been configured for true dual port access, meaning two independent lookups can be performed each clock cycle. They are also configured to be 2^{14} bits wide, the maximum full address space of the onboard 18Kb Block RAMs. If necessary in the future, these can be reconfigured to 2^{15} bits wide and use two 18Kb Block RAMs to make a single 36Kb Block RAM in order to lower the false positive rate. The Block RAMs also have their contents initialized at compile time rather than loading them after the system is running in order to simplify the design. The Bloom filter contents is generated in Matlab [Mat14] based on enumerated valid commands.

3.3.2 Current Sense Circuit.

The current sense circuit is a key element in determining whether or not the PLC actually executed a command sent to it. Its operation is based on Ohm's Law, shown in Equation 3.1. The voltage drop V across a circuit element is

$$V = I * R \tag{3.1}$$

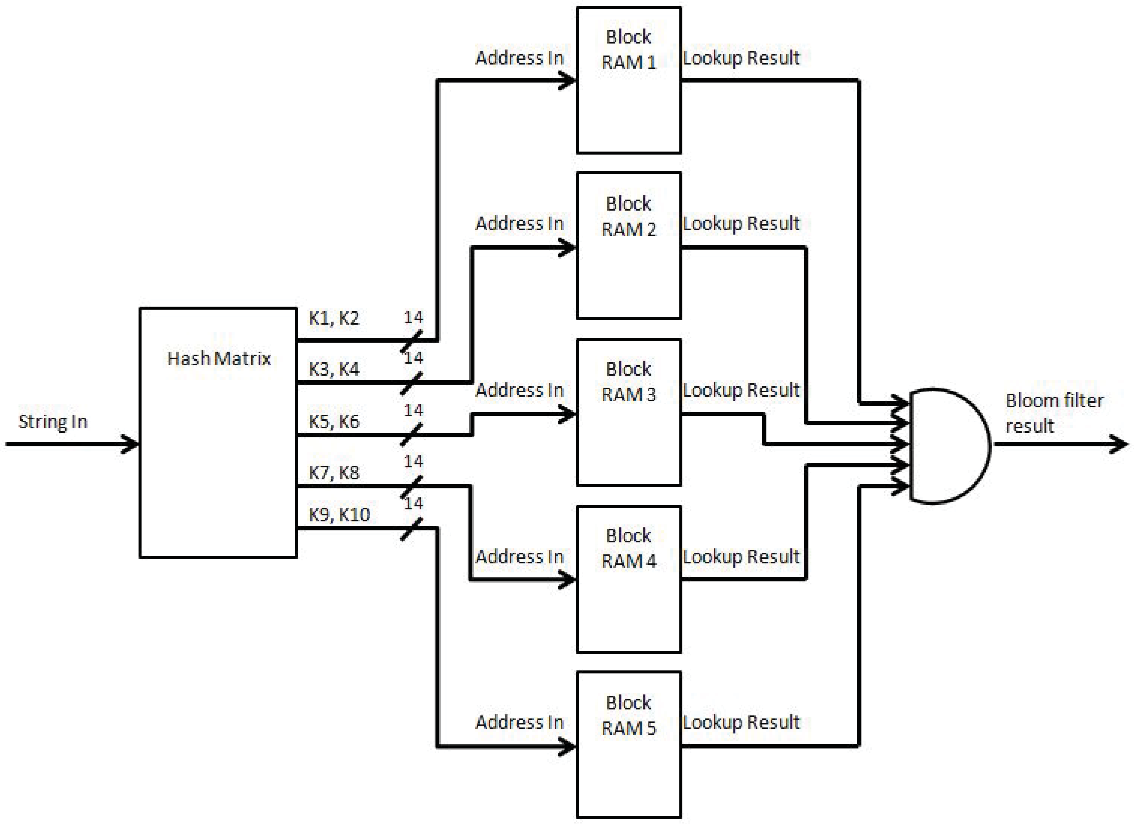


Figure 3.9: Simplified Bloom Filter Block Diagram

where I is the current flowing through the element and R is the resistance of the element. Since it is known that the current loop is restricted to a range between 4 mA and 20 mA, all that remains is to use a circuit element with a known resistance. Figure 3.10 depicts the current sense circuit with a 200 Ω current sense resistor. The resistance of 200 Ω was chosen to provide a 0.8 V to 4 V range for the ADC to sense. If deployed with a real PLC, the current sense resistor would be a much smaller resistance, likely in the 0.1 mΩ to 1 mΩ range, in order to minimize power dissipation. A resistor in that range is not used for this research due to the necessity of designing signal amplification circuitry and custom printed circuit boards.

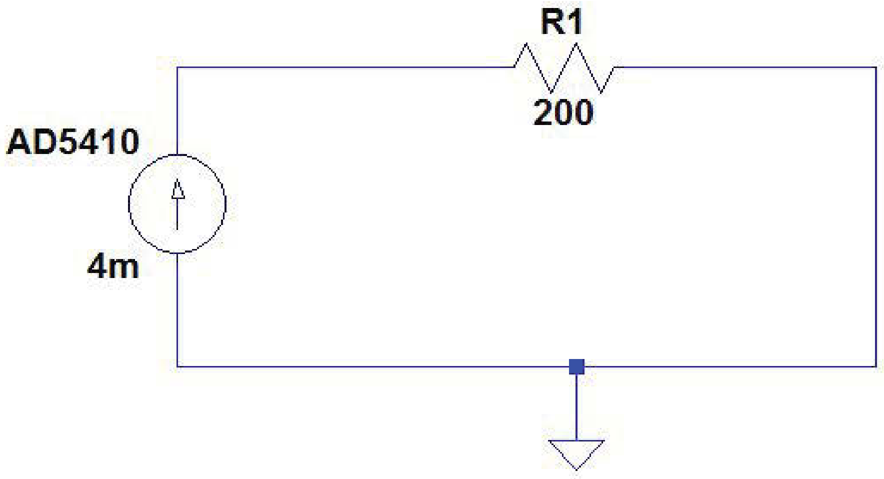


Figure 3.10: Current Sense Circuit

3.3.3 ADC Bank.

The ADC Bank is a structure that provides a single point to interact with all of the ADCs. The bank allows for control of up to five ADCs, each set to monitor a different line. Figure 3.11 shows a simplified block diagram of the ADC Bank.

The ADC selected for use in this design is the ADS7818p from Texas Instruments primarily due to its ability to perform 500,000 conversions per second. The ADS7818p is a 12-bit ADC, meaning it should be able to distinguish between 2^{12} different voltage levels between 0 V and twice its reference voltage [Cor98]. For the SIEVE system, the ADCs are set to use their internal 2.5 V internal reference, giving them an input voltage range from 0 V to 5 V. This means that the difference between a reading of 0 and 1 (or 4 and 5, 512 and 513, and so on) is 1.22 mV. Unfortunately, the 12-bit resolution of the ADCs restricts the SIEVE system to eight-bit analysis. Since MODBUS registers are 16 bits wide, the ADC cannot distinguish all of the possible values that a register can take. The

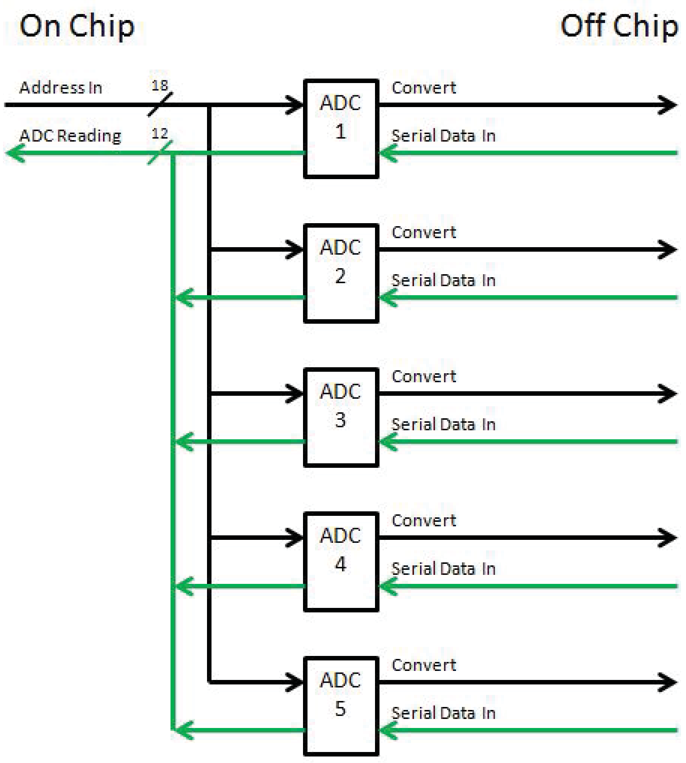


Figure 3.11: Simplified ADC Bank Block Diagram

range of analysis is further narrowed from 12 bits because a line whose data value is set to zero is still outputting 4 mA or 0.8 V in the current sense circuit, not zero mA. This is not a flaw, it is simply the nature of current loop control and must be taken into consideration in the design process. From this, the number of levels that the ADS7818p should be able to distinguish is 3.2 V (4.0 V - 0.8 V) divided by 1.22 mV or 2,621 different levels. This range is larger than 2^{11} but less than 2^{12}. Thus, since 16-bit and 12-bit data is not able to be handled, the eight-bit data range was chosen.

The ADS7818p converts an analog voltage to a digital reading over the course of 16 clock cycles. The conversion takes that much time due to the chip outputting the reading serially then requiring time to reacquire the analog voltage level. Its maximum clock rate is 8 MHz [Cor98], much slower than the system clock rate of 125 MHz. Since a conversion takes 16 clock cycles and the length of an 8 MHz clock cycle is 125 ns, the convert time is 2000 ns or 2 μs. Taking the inverse of this conversion time shows that the maximum number of conversions per second is 500,000 or 500 KHz, the ADS7818p's maximum rating. This derivation shows how the number of conversions per second is based off of the chip's clock rate. Unfortunately, the 125 MHz main system clock rate does not evenly divide down to 8 MHz. This means the supplied clock rate will be slightly slower than 8 MHz, increasing the amount of time needed to convert an analog voltage. The slow clock was derived by toggling a signal up or down every 8 main system clock cycles. At 125 MHz, a clock cycle is 8 ns long. The derived clock rate is then

$$f_{derived} = \frac{1}{16 * 8\ ns} = 7.8125\ MHz \tag{3.2}$$

This equates to a conversion time of 2.048 μs or 488,281 conversions per second.

A conversion is triggered when the address_in signal of the ADC (shown in Figure 3.11) is equal to the address of the line it is monitoring. Each ADC monitors a unique address corresponding to a specific monitored line. It is up to the operator to ensure that the monitored addresses are unique and that only one ADC is triggered at a time. When the ADC is triggered, the serial data is stored in the lowest bit of a shift register and gradually shifted to the left. After 16 slow clock cycles, a finished signal goes high and the digital reading is presented to the CPU in a register.

Since reading the current setting of the line is important, a pilot study was conducted to determine how accurately the ADC can measure the current. This was accomplished using two ML507 FPGA boards, one running the ADC bank (board A) and the other running a 12 bit Digital to Analog Converter (DAC) current source (board B). After setting DAC current source value to zero (4 mA), the ADC is queried to determine the voltage across the current sense resistor. Next, board A signals board B to increment the DAC current source value by 1. Five milliseconds later, the line is measured again. This continues until the line value wraps around back to zero. This experiment is conducted five times. Figure 3.12 shows the averaged readings across those five experiments. It is clear that the ADC at least matches the expected upward trend; however, it appears to fail to provide readings that distinguish adjacent data values. The worst run of the same average value is from data values 171 to 202, totaling to 31 values with the same readings. The ADC also reads higher voltages than expected – 0.3 V for lower data values and 0.16 V for higher values. These inaccuracies are likely due to a circuit grounding issue..

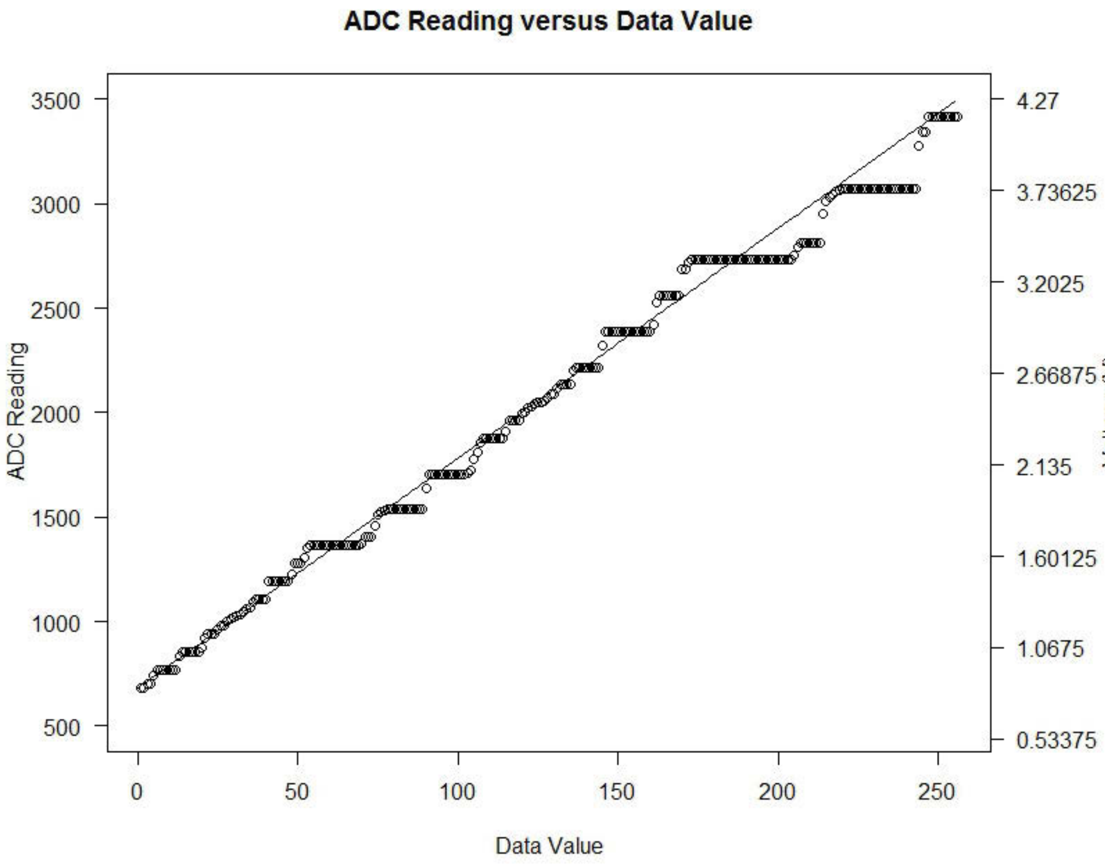


Figure 3.12: ADC Readings for Each Data Value

A linear model of the ADC readings was calculated using R [fSC14] and found to be

$$y = 682 + 10.77 * x \tag{3.3}$$

where x is the eight-bit data value. The linear model has an r^2 value of 0.99, indicating statistical significance. This model is used in software to transform the ADC readings to an approximation of the data value of the line.

3.3.4 Software.

The custom software developed for the SIEVE system is responsible for operating the developed hardware structures, maintaining the algorithm state (i.e. a response is expected, processing commands and responses, and logging invalid packets/sending warnings). It is written in C and runs on the embedded PowerPC CPU.

Operation of the developed hardware structures is supported through driver files that are automatically generated by the Xilinx software. Interaction with the structures is established using 32-bit hardware registers. Each device uses four registers for data input, output, status updates, and command and control. Table 3.3 shows the software registers and their purposes for both the MODBUS Inspection Core and the ADC Bank. After issuing a command to begin operation, stable results are obtained by continuously polling the device status registers until they indicate the operation is complete.

Maintaining the algorithm state is done through the use of two flags: command_expected and response_expected. The algorithm begins operation expecting a command to the PLC. If a valid command is received, command_expected is set to zero and response_expected is set to one. Then, if a valid response from the PLC is received, response_expected is set to zero and command_expected is set to one. This dual flag setup provides a means to detect communication that does not occur as expected such as a command that is received before a response is received for the previous command.

Table 3.3: Software Registers

Device	Register	Use
MODBUS Inspection Core	0	Four byte packet in
	1	Packet size
	2	Device commands
	3	Device status
ADC Bank	0	Address to query
	1	Address valid
	2	Device status
	3	Query result

Command and response processing are the primary purposes of the developed software. Since the Bloom filters have the possibility of failing to detect an invalid command due to the false positive problem, further scrutiny is required for validation. This is accomplished by inspecting the function code, base address, and in the case of writing to holding registers, checking if the data is within the valid range. This is accomplished with lookups in three files stored in the ML507's SRAM module. The SRAM module has 1 MB of storage and faster access times than the DDR2 SDRAM. After a command has been returned as a valid from the MODBUS Inspection Core, the software takes over inspection. First, the function code is extracted in order to determine how to inspect the remaining fields. Once the correct routine is determined, a lookup is performed based on the data construct that the command is operating on and the base address to find which actions can be performed at that address. For read commands, inspection stops here as a read at an address is either allowed or not allowed. Write commands undergo further inspection. For coils, only the Write Single Coil command has further inspection because its data field is defined as either being 0xFF00 or 0x0000. If the data field is anything other than those two values, the command is determined to be suspicious. Further inspection is not necessary for Write Multiple Coils because the data is packed as single bits rather than two bytes. The bits can only be one of two values, both of which are valid for coils. For holding registers, two lookups are performed to determine the lowest value that can be written and the highest value that can be written. If the data is out of this range, the command is determined to be suspicious.

Response processing is simpler than command processing yet more difficult to accomplish. The main issue with processing MODBUS/TCP responses is that they do not always have all of the information needed to properly interpret them. Table 3.4 shows the response formats for the commands implemented for the SIEVE system. Of the eight commands, only two responses (Write Single Coil and Write Single Holding Register) contain enough information for processing without consulting the command. Thus, after a valid command has been processed it must be stored for use when processing the response. In all cases, the base address for inspection and number of operations are loaded from the previous command. Next, a counter is set up to iterate over all addresses affected by the command. If one of those addresses matches a monitored address, a list of which is stored in the program, the ADC connected to that address' corresponding line is queried. The result returned, however, is a 12-bit encoding of the voltage drop across the 200 Ω current sense resistor. Using the linear model found in Section 3.3.3 the returned result is converted to an approximation of the original data value. However, due to the inaccuracies of the ADC as shown in Section 3.3.3, a range of values must be considered for the calculated data value. Given that the worst run equivalent values was determined to be 32 values long, as long as the calculated value is within ±15 of the command's data value for writes or response's data value for reads, the response is accepted.

Packets are logged using the libpcap format. This file format is readable by both Wireshark [Wir14b] and tcpdump [Tcp14], two widely-used packet capture programs.

Packets are stored in a binary format and are timestamped with a 32-bit seconds field and a 32-bit microseconds field [Wir14a]. The log file is maintained in the off FPGA DDR SDRAM due to the large storage capacity (256 MB) it offers. When an operator wants to inspect the log file, it is copied to a compact flash card that can be removed and read by another computer. Reporting that a packet has been logged due to being suspicious is accomplished by sending a text message across RS232 that is read on another user operated computer.

Table 3.4: Response Formats

Function Code	Command	Response Format
0x01	Read Coils	Byte count followed by data
0x02	Read Discrete Inputs	Byte count followed by data
0x03	Read Holding Registers	Byte count followed by data
0x04	Read Input Registers	Byte count followed by data
0x05	Write Single Coil	Echo of command
0x06	Write Single Register	Echo of command
0x0F	Write Multiple Coils	Base address written followed by number of coils written
0x10	Write Multiple Holding Registers	Base address written followed by number of registers written

IV. Methodology

This chapter covers the methodology used to test and evaluate the SIEVE system. Section 4.1 covers the goals and hypothesis of the research, Section 4.2 covers the boundaries of the System Under Test (SUT), Section 4.3 covers the services the SIEVE system provides, Section 4.4 covers the workloads used to drive the SUT, Section 4.6 covers the parameters of the SIEVE system, Section 4.7 covers the factors varied during the evaluation of the SIEVE system, Section 4.8 covers the evaluation techniques of the experiment, Section 4.9 covers the experimental design, and Section 4.10 summarizes the experiment.

4.1 Goals and Hypothesis

The goal of this research is to evaluate the effectiveness of the SIEVE system that verifies the integrity of incoming traffic to a PLC and the integrity of the PLC's operations. The success of this system will be evaluated like that of an intrusion detection system. It is evaluated by its ability to successfully verify the integrity of the network traffic and device actions. A system that fails to perform these actions is inadequate. In SCADA applications, this service cannot impact normal device operation and needs to be fast enough to identify an action (incoming command or device response) that is not valid as soon as it happens. The hypothesis of this research is that the SIEVE system is able to verify the integrity of 100% of incoming network traffic and PLC responses in realtime, that is complete inspection and, if necessary, logging of packets in 5 ms or 200 command per second frequency. The 5 ms requirement is derived from the specifications of Allen-Bradley PLCs in [Roc12]. The most common analog scan time for these PLCs is 10 ms. Halving that scan time as a goal results in a goal inspection time of 5 ms. The secondary goal of the research is also to determine whether Bloom filters provide any benefits in identification of invalid commands. Since Bloom filters only return negative lookup results for commands that are not inserted into them, the hypothesis for this goal is that for invalid MODBUS/TCP commands a well tuned Bloom filter will reduce the amount of time needed for processing.

4.2 System Boundaries

The SUT, also referred to as the Remote Device Integrity Verification System (RDIVS), consists of an emulated PLC and the SIEVE system. Figure 4.1 shows the block diagram of the SUT. In the RDIVS, the emulated PLC reports or modifies its state based on commands received from the MTU – performing basic SCADA operations. The use of an emulated PLC instead of a real one is covered in Section 4.8 and Design and operation of the emulated PLC is covered in Appendix B. The SIEVE system is the Component Under Test (CUT) and implements the specifications-based integrity verification algorithm. The SUT does not include the MTU or devices being controlled or read from.

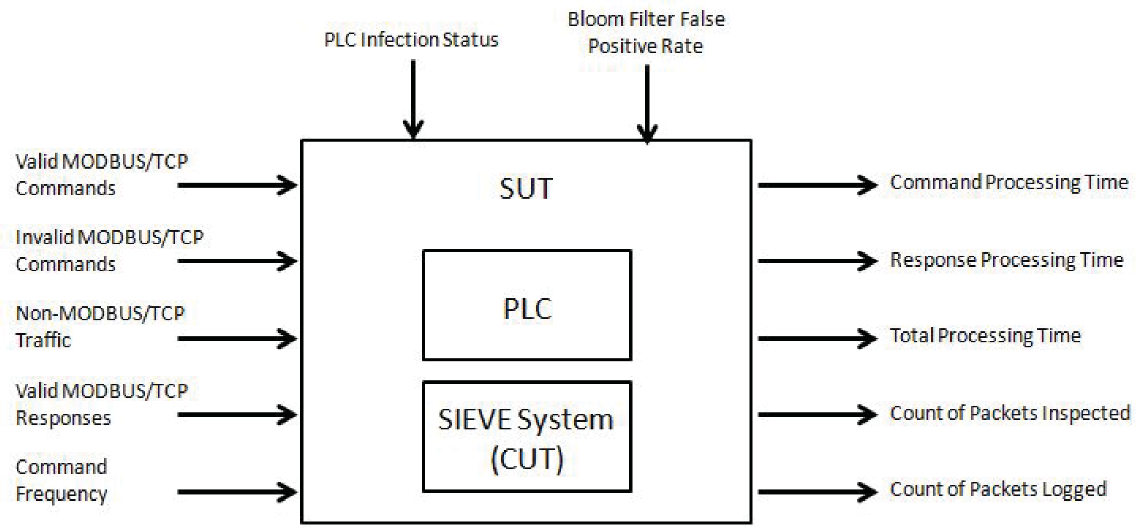


Figure 4.1: Remote Device Integrity Verification System (RDIVS)

4.3 System Services

Integrity verification is the primary service provided by the RDIVS. This service takes the commands sent to the system as input and responds based on the change in state of the PLC. Two outcomes are identified for this service, and they are doing nothing or sending an alert. Doing nothing occurs when the received commands are correctly replied to by the PLC, in this case changing system state or truthfully responding its state. Sending an alert happens when the PLC does not reply correctly to the received command (e.g., changing to an incorrect state or falsifying its state to the MTU). Each outcome can also be invoked through improper behavior of the CUT.

4.4 Workload

The workload to the system is a series of commands from the MTU to the PLC and responses from the PLC to the MTU. Three workloads are generated: a valid MODBUS/TCP workload, an invalid command MODBUS/TCP workload, and a non-MODBUS/TCP workload. The commands are sent to the system via Ethernet in MODBUS/TCP packets at regular intervals, mimicking the characteristics of a SCADA network.

The primary characteristics of the workload is the command types, their length, and the interval at which they are sent. Command type is chosen as it ties directly into the complexity of the CUT in that implementing more commands means more hardware structures and perhaps more time to interpret them. In an active SCADA network, the majority of these commands are reads from or writes to coils and registers. Command length is how many operations are performed per command. This affects performance in that each operation must be scrutinized in order to determine validity, thus more operations means a higher processing time. Lastly, the interval between packets also has a large impact on the CUT. Since all packet inspections and ADC queries must happen between command packets, if command packets arrive faster than the SIEVE system can process them the receive buffers could fill up. If a packet arrives when the receive buffers are full, it is dropped. If 100% of packets cannot be inspected, the CUT fails the test.

4.4.1 Valid MODBUS/TCP Workload.

The valid MODBUS/TCP workload consists of MODBUS/TCP commands that are well defined for the PLC. This means that all of the commands sent in this workload can be executed correctly and without error by the PLC. Table 4.1 lays out the commands and their lengths. Each command and response was generated using Triangle MicroWorks' SCADA Test Harness program [TM14]. Read Discrete Inputs and Read Input Registers commands and responses are not included in the workload because they are functionally identical to Read Coils and Read Holding Registers.

Table 4.1: Valid Commands and Responses

Command	Function Code	Number of Operations
Read Coil	0x01	Minimum (1)
		Maximum (2000)
Read Holding Register	0x03	Minimum(1)
		Maximum(125)
Write Single Coil	0x05	1
Write Single Holding Register	0x06	1
Write Multiple Coils	0x0F	Minimum (1)
		Maximum (1968)
Write Multiple Holding Registers	0x10	Minimum(1)
		Maximum(123)

4.4.2 Invalid MODBUS/TCP Workload.

The invalid MODBUS/TCP workload consists of all of the commands from the valid workload except modified to be invalid for the PLC. This means that execution of the command is not defined for the PLC, that is, it will return a MODBUS Exception Response PDU rather than a MODBUS Response PDU, or execute an undefined action. The commands are made to be invalid by modifying the base address field to be an invalid address for the emulated PLC. This was done using a hexadecimal editor and confirmed in Wireshark [Wir14b]. Modifying the base address of the command allows for fine tuning of the amount of processing needed to determine that the command is invalid. For commands that perform one operation, such as Write Single Coil, the invalid address is the first and only operation inspected. For the maximum length commands, such as Read Coils with a length of 2000, the invalid address could be the first address operated on, the last address operated on, or somewhere in the middle. Table 4.2 lays out the workload by command, number of operations, and when the invalid address will be invoked.

4.4.3 Non-MODBUS/TCP Workload.

A non-MODBUS/TCP workload for the SUT could be made up of any number of different packet types as long as they are not MODBUS/TCP. For a more real world approach, a port scan of a PLC was determined to be suitable for this workload. A problem with this though is that the emulated PLC used in the SUT does not have an IP stack. Thus the non-MODBUS/TCP workload is a Zenmap [Mar14] port scan of an Omron CP1L PLC that was recorded using Wireshark [Wir14b] and replayed using Tcpreplay [Tur14]. Every packet in this workload is undefined by definition for the emulated PLC. Zenmap was setup to perform a default port scan of the PLC with the command

```
nmap -T4 -A -v
```

The first 50 synchronization requests and responses are used for the workload.

Table 4.2: Invalid Commands and Responses

Command	Number of Operations	Invalid Location
Read Coil	Minimum (1)	N/A
	Maximum (2000)	Best (1)
	Maximum (2000)	Worst (2000)
Read Holding Register	Minimum(1)	N/A
	Maximum(125)	Best (1)
	Maximum(125)	Worst (125)
Write Single Coil	1	N/A
Write Single Holding Register	1	N/A
Write Multiple Coils	Minimum (1)	N/A
	Maximum (1968)	Best (1)
	Maximum (1968)	Worst (1968)
Write Multiple Holding Registers	Minimum(1)	N/A
	Maximum(123)	Best (1)
	Maximum(123)	Worst (123)

4.4.4 Command Frequency.

The command frequency dictates how quickly the CUT must process both the command packet and response packet. Since the SIEVE system has a finite amount of memory to store received packets, it must process them fast enough such that packets do not drop. Four frequencies are used: 200 commands per second, Valid Operation Break, Invalid Command - Saturated Break, and 5000 commands per second. The first frequency, 200 commands per second, corresponds to a 5 ms interval between packets and is the rate at which the SUT must be able to inspect and log all packets. Valid Operation Break is a command frequency which is higher than 200 commands per second and causes the SUT to process the Valid MODBUS/TCP workload incorrectly. Invalid Command - Saturated Break is a command frequency which is higher than Valid Operation Break that causes the SUT to incorrectly process the Invalid MODBUS/TCP workload when the Bloom filters are saturated. Lastly, 5000 commands per second is the fastest that the emulated PLC is able to respond correctly to MODBUS/TCP commands.

4.5 Performance Metrics

Metrics recorded for the SUT are command processing time, response processing time, total processing time, count of packets inspected, and count of packets logged. The first three metrics are used to determine how long it takes the CUT to process a packet while the latter two valid the correct and accurate operation of the CUT.

The command processing time metric is the time it takes the CUT to completely process a packet sent to the PLC, while response processing time is the time it takes the CUT to completely process a packet sent from the PLC. For both metrics, time begins upon complete receipt of the packet by the CUT. This is determined by the receive FIFO of the CUT reporting that a packet has been received. Time also ends for both metrics upon complete verification of the validity of the packet or logging the packet and sending a warning. The difference between the two metrics is the type of packet being processed.

The third metric, total processing time, is derived from the command processing time and response processing time. It is the addition of the two aforementioned metrics for corresponding commands and responses and represents the total amount of time required by the CUT to process a conversation between the MTU and the PLC. Total processing time, $t_{processing\ time}$, is defined as

$$t_{processing\ time} = t_{command} + t_{response} \tag{4.1}$$

where $t_{command}$ is the command processing time and $t_{response}$ is the response processing time.

The count of packets inspected and count of packets logged are both measures of the CUT's inspection abilities. The count of packets inspected is incremented every time a packet is inspected by the CUT whereas the count of packets logged is incremented only when a packet is logged. The first measures the system's ability to process packets in a timely fashion while the second measures the system's ability to discern valid packets from invalid packets. If the count of packets inspected is less than the number of packets actually sent, then the CUT is dropping packets. For the Valid MODBUS/TCP workload the count of logged packets should be equal to zero since every packet is well defined for the emulated PLC. For all other workloads, the count of logged packets should be equal to the total number of packets sent since every packet should be logged.

4.6 System Parameters

The following parameters affect system performance.

4.6.1 Communication Protocol.

The communication protocol between the SUT and MTU is the language that both devices speak to each other. Since every communication protocol utilizes different arrangements of bits in the control packets, interpretation can be more or less complex depending on the implementation. The communication protocol used in the experiment is MODBUS/TCP.

4.6.2 Communication Medium.

The communication medium between the SUT and MTU is the means by which the two communicate. The medium implemented can greatly affect network latency as some are slower than others. Mediums capable of higher transmission speeds mean more time is available to update the system image and verify PLC output. The communication medium used in the experiment is Ethernet.

4.6.3 MODBUS Functions Implemented.

Depending on how a PLC is used in a SCADA network, different commands will produce different results. Depending on the PLC model, some commands may not be implemented at all and have an undefined effect on the system. This parameter affects performance in that the number of functions implemented makes the inspection hardware and software more complex. The emulated PLC implements all of the MODBUS/TCP commands shown in Table 4.1. It has 2000 discrete inputs, 2000 coils (addresses 0x0000 - 0x07CF) and 125 holding registers and input registers (addresses 0x0000 - 007C) defined. All of the aforementioned addresses are valid for reading and writing using commands that perform multiple operations (e.g. Write Multiple Coils). Commands that perform one operation only, such as Write Single Coil, are valid only for address 0x0000 for coils and holding registers. Holding registers are also configured for two data values only: 20 and 120. The total number of valid read commands is 4250 (2000 Discrete Inputs + 2000 Coils + 125 Holding Registers + 125 Input Registers) and the total number of valid write commands is 4254 (2*(2000 Coils) + 2*(125 Holding Registers) + 2*(1 Coil) + 2*(1 Holding Register)).

4.6.4 FPGA.

The FPGA upon which the IDS is implemented has limits according to its clock speed, microprocessor used, and amount of custom hardware that can be implemented. As the virtual system model becomes more complex, more FPGA fabric will need to be used to implement the custom hardware. If the system becomes too complex, the model may not be maintainable by the FPGA. The FPGA used in the CUT is a Xilinx XC5VFX70T.

4.7 Factors

The following are different factors and their levels used to evaluate the performance of the specifications-based integrity verification algorithm. Tables 4.3, 4.4, and 4.5 lay out the factors and levels for each of the three experiments.

4.7.1 Number of Lines Monitored.

The number of lines monitored affects how many ADC queries are required. The two levels identified are the minimum number of monitored lines (1) and maximum number of monitored lines (5) for the CUT.

4.7.2 Infection Status of PLC.

The infection status of a PLC will indicate whether commands are followed or requests for data falsified. In the case of an infected PLC, alerts would be expected to be raised based on PLC responses. The two levels are not infected and infected.

4.7.3 Infected Line Address Location.

The infected line's address location affects performance for commands that perform multiple operations. The best location for a command would be the first address checked whereas the worst location would be the last address in a series of operations. The two levels are best and worst.

4.7.4 Bloom Filter False Positive Rate.

The Bloom filter false positive rate will determine whether an invalid command is caught in hardware or software. Since the rate differs based on the hash functions used, commands implemented for the PLC, and size of the block RAM components, two levels are identified. The first level corresponds to a properly configured Bloom filter whose false positive rate is not equal to 100%, referred to as non-saturated. Using the command counts derived in Section 4.6.3 (n=*4250 or 4254*), the Bloom filter widths (m=2^{14}) and number of hash functions (k=*10*) from Section 3.3.1.7, and Equation 2.1, the non-saturated false positive rate for the read command Bloom filter is 2.11% and 2.14% for the write command Bloom filter. The second level corresponds to a saturated Bloom filter which reports all queries as being present. This was accomplished by setting all entries in both the read command and write command Bloom filters to '1'.

Table 4.3: Experiment 1 Factors and Levels

Factor	Level 1	Level 2	Level 3
Workload	Valid MODBUS/TCP	Invalid MODBUS/TCP	Non MODBUS/TCP
Bloom Filter False Positive Rate	Non-saturated	Saturated	

Table 4.4: Experiment 2 Factors and Levels

Factor	Level 1	Level 2
PLC Infection Status	Not Infected	Infected
Infected Location	Best	Worst
Number of Lines Monitored	1	5

Table 4.5: Experiment 3 Factors and Levels

Factor	Level 1	Level 2	Level 3	Level 4
Workload	Worst Case Valid Command and Response	Worst Case Invalid Command	Worst Case Infected Response	Non-MODBUS/TCP
PLC Infection Status	Not Infected	Infected		
Bloom Filter False Positive Rate	Non-saturated	Saturated		
Command Frequency	200 Commands per Second	Valid Operation Break	Invalid Command Saturated Break	5000 Commands per Second

4.8 Evaluation Technique

To evaluate the system direct measurement is used with PLC emulation. No analytical model exists for this application, and a simulation would be difficult to create for such a complex system. Testing the system in an emulated environment should provide results reflective of an active environment. Validation of the integrity verification capability of the system is accomplished by using either a purely valid workload or purely invalid workload, in which case either all packets are not logged or logged. This ensures all input types are investigated.

PLC emulation is chosen as an alternative to using an actual PLC due to the infected PLC factor. Achieving the infected factor level is difficult with an actual PLC because PLCs cannot falsify their line status without being legitimately infected by malware such as Stuxnet. To do so could result in permanent damage to the device and would require precise control over how the malware operates. Emulation is achieved using an FPGA development board running a simple software loop that drives the DACs and sends a response from a list based on the command it receives. It will perform no other processing.

Figure 4.2 shows a block diagram of the experimental setup and consists of the following:

- 1 Cisco 24-port gigabit Ethernet Switch (model WS-C3560G-24PS-S) configured with two SPAN ports.
- 1 HP 8570W laptop running Backtrack 5 R3 and Tcpreplay acting as the MTU, connected to a normal port on the switch.
- 1 Xilinx ML507 Board Revision A Virtex 5 development board acting as the emulated PLC, connected to a normal port on the switch.
- 5 AD5410 Digital to Analog Current Source ICs to act as inputs or outputs to the emulated PLC.

- 1 Xilinx ML507 Board Revision A Virtex 5 development board as the CUT, connected to a SPAN port on the switch.

- 5 ADS7818 Analog to Digital Converter ICs. These chips are used by the CUT to monitor the PLCs inputs or outputs.

- Asus G73J laptop running Windows 7. This laptop uses two RS232 connections to monitor the state of the CUT and emulated PLC and is also connected to the other SPAN port on the switch in order to monitor the experiments.

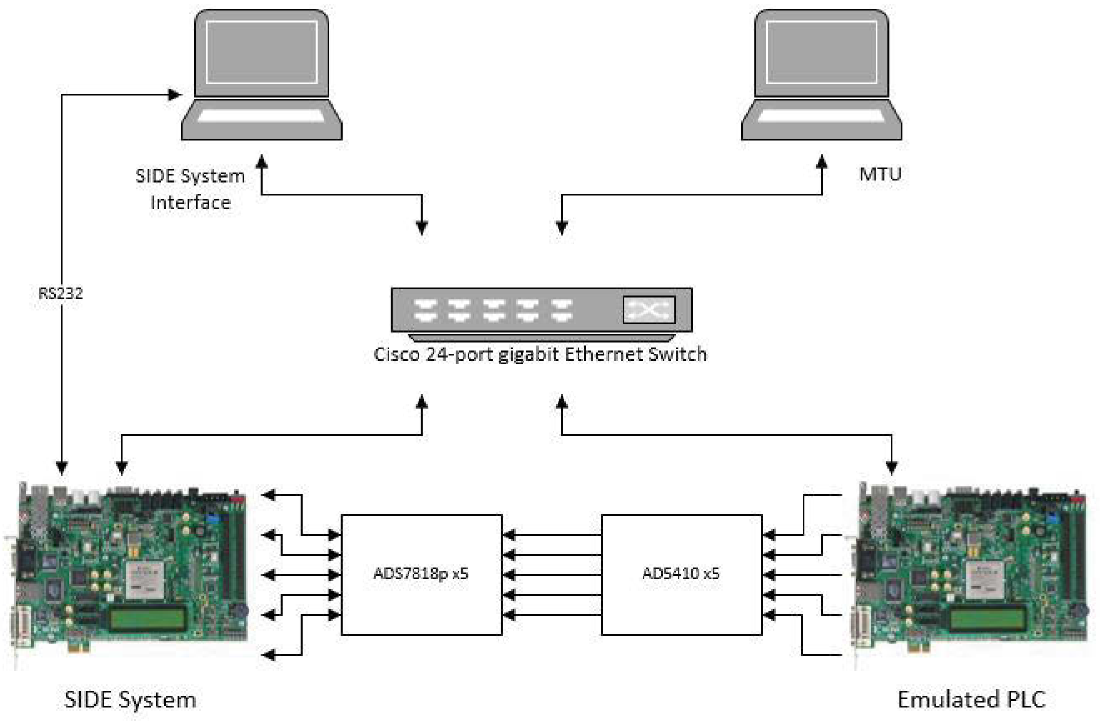


Figure 4.2: Experiment Block Diagram

A photo of the actual experimental setup is shown in Figure 4.3.

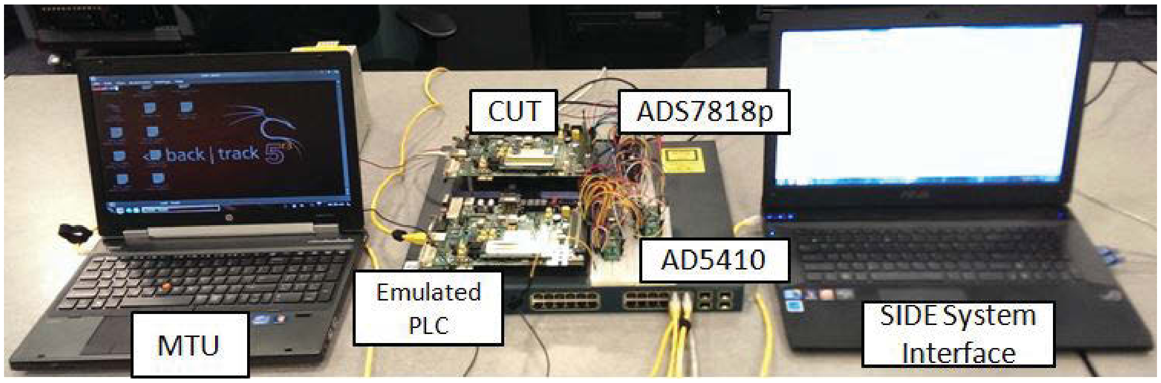


Figure 4.3: Experimental Setup

4.8.1 Experiment 1.

The purpose of experiment 1 is to determine packet processing time for command packets. In this experiment, 50 packets of each command type from each workload in Section 4.4 are sent to the emulated PLC from the HP 8570W using Tcpreplay [Tur14]. The commands are sent at a rate of 200 packets per second, corresponding to one command every 5 ms. As the commands are sent, the command processing time is recorded by the CUT. Sending 50 packets was determined to provide sufficiently small confidence intervals. Additionally, five commands are sent prior to recording the command processing time in order to “warm up” the board by caching instructions and necessary data.

4.8.2 Experiment 2.

Experiment 2 is accomplished in much the same way as Experiment 1. In this case, however, only the valid MODBUS/TCP command workload is used. Commands are again sent to the emulated PLC using Tcpreplay [Tur14] at a rate of 200 commands per second, however response processing time is recorded instead of command processing time. The infected PLC factor level is achieved by modifying the emulated PLC to drive the DACs to a value that does not match the received command.

4.8.3 Experiment 3.

The goal of experiment 3 is determine the SIEVE system's ability to process packets based on the worst case total processing time from each workload and command frequency. The count of packets inspected and count of packets logged are recorded for this experiment. The worst case total processing time is determined based on the results of Experiments 1 and 2. The command frequencies chosen are also based off of the results of Experiments 1 and 2 and should cause the CUT to not perform optimally. The experiment is run by sending 5000 commands to the emulated PLC which should in turn reply with 5000 responses, resulting in a total of 10000 packets being examined. The workloads will either be purely invalid or valid, meaning all packets should logged or not logged.

4.9 Experimental Design

The first experiment is a partial factorial design and uses all four workloads specified above and varies the bloom filter false positive rate for the invalid MODBUS/TCP workload and non-MODBUS/TCP workload. The false positive rate is not varied for the valid workload because the commands pass through no matter what. The experimental scenarios used for this experiment are listed in Appendix A. Three replications are performed in order to reduce the variance in the data. The total number of trials is therefore 3 repetitions*(50 commands*(10 valid commands+14 invalid commands*2 bloom filter levels + 1 non-MODBUS/TCP*2 bloom filter levels))=6000. A one variable t-test is performed in order to determine the mean command processing time, the standard deviation, standard error, and 95% confidence interval.

The second experiment is a partial factorial design and uses the valid MODBUS/TCP command workload. The factors varied will be the number of lines monitored and the infection status of the PLC totaling to 36 scenarios (listed in Appendix A). The total number of trials is 3 repetitions * 50 packets * 36 scenarios = 5400. A one variable t-test is performed in order to determine the mean response processing time, the standard deviation, standard error, and 95% confidence interval.

The third experiment is a partial factorial design and uses the worst case exchanges for each workload type and varies command frequency. This should determine how the CUT responds as command frequency increases according to the workload. The experiment will consist of 3 repetitions * 10000 packets * 4 command frequencies * 4 workloads * 2 Bloom filter false positive rates = 960000 trials. A one variable t-test is performed in order to determine the 95% confidence intervals for the count of packets intercepted and the count of packets logged.

4.10 Methodology Summary

SCADA networks are involved in many critical infrastructure applications. As these networks are increasingly connected to the Internet, more and more opportunities come into existence for exploitation. Protection of these networks is difficult due to their unique operating behaviors and limited resources, necessitating a new approach to defense. The purpose of this research is to investigate a specifications-based integrity verification technique to defend these vulnerable networks. The expected result is that this technique is scalable to the system and can operate within realtime constraints.

This chapter introduces the methods by which the experimental goal is fulfilled. The bounds of the system are established and performance metrics introduced. Three partial factorial experiment are introduced that determine command processing time, response processing time, total processing time, count of packets inspected, and count of packets logged.

V. Results

THIS chapter presents the results and analyses of the three experiments. Section 5.1 covers the results of Experiment 1. Section 5.2 covers the results of Experiment 2. Lastly, Section 5.3 covers the results of Experiment 3.

5.1 Results and Analysis of Experiment 1

This section presents the results of experiment 1. Results are presented both in tabular format with specific notation to denote the measured command. First the command type is stated, then command length if applicable, and lastly invalid address location if applicable. An example for a maximum length valid Read Coils command is Read_Coils_Max. An invalid maximum length Read Coils command with the invalid address in the worst location is denoted Read_Coils_Max_Worst.

5.1.1 Valid MODBUS/TCP Workload.

Table 5.1 details the results of the one-variable t-test performed on the valid MODBUS/TCP commands that operate on coils. The table contains the number of packets sent, the mean CPU cycles, the standard deviation, the standard error of the mean, and the 95% confidence interval. The table is ordered by mean CPU cycles, equating to command processing time. The range of mean command processing time for commands that operate on coils is 2528.893 to 99550.633 cycles.

Table 5.2 details the results of the one-variable t-test performed on the valid MODBUS/TCP commands that operate on holding registers. The table contains the number of packets sent, the mean CPU cycles, the standard deviation, the standard error of the mean, and the 95% confidence interval. The table is ordered by mean CPU cycles, equating to command processing time. The range of mean command processing time for commands that operate on holding registers is 2256.947 to 23897.107 cycles.

Table 5.1: Valid Command Processing Times - Coils

Command Type	Packets Sent	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
Write_Single_Coil	50	2528.893	1.777	0.145	(2528.607, 2529.180)
Read_Coils_Min	50	2551.307	2.838	0.232	(2550.849, 2551.765)
Write_Multi_Coil_Min	50	2629.660	4.096	0.334	(2628.999, 2630.321)
Read_Coils_Max	50	92509.953	41.791	3.412	(92503.211, 92516.696)
Write_Multi_Coil_Max	50	99550.633	90.833	7.417	(99535.978, 99565.288)

Table 5.2: Valid Command Processing Times - Holding Registers

Command Type	Packets Sent	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
Read_HR_Min	50	2556.947	1.809	0.148	(2556.655, 2557.239)
Write_Single_HR	50	2582.213	10.369	0.847	(2580.540, 2583.886)
Write_Multi_HR_Min	50	2726.927	8.407	0.686	(2725.570, 2728.283)
Read_HR_Max	50	8271.887	3.519	0.287	(8271.319, 8272.454)
Write_Multi_HR_Max	50	23897.107	31.039	2.534	(23892.099, 23902.115)

Overall, for the valid MODBUS/TCP workload it becomes clear that minimal length commands, that is performing one operation, all have very similar command processing times. Maximum length write commands, for both coils and holding registers, both require more time than their maximum length read counterparts. This is despite maximum length reads performing more operations (2000 operations compared to 1968 operations for coils). For coils this is likely due to a combination of the additional Produce Mini Packets state in the MODBUS Inspection Core combined with the fact that the maximum length Write Multiple Coils command is 313 bytes wide and takes longer to copy across the system than the 66 byte maximum length Read Coils. For holding registers it is likely due more to the overhead of two additional memory lookups for the data range than copy latencies. Differences between maximal length commands for coils and holding registers is due to the sheer number of operations performed in coil commands compared to holding registers (2000 reads for coils and 125 reads for holding registers).

5.1.2 Invalid MODBUS/TCP Workload.

Table 5.3 details the results of the one-variable t-test performed on the invalid MODBUS/TCP commands that operate on coils in a system with non-saturated Bloom filters. The table contains the number of packets sent, the mean CPU cycles, the standard deviation, the standard error of the mean, and the 95% confidence interval. The table is ordered by mean CPU cycles, equating to command processing time. The range of mean command processing time for commands that operate on coils is 1799.167 to 7853.620 cycles.

Table 5.3: Invalid Command Processing Times - Non-saturated Bloom Filter - Coils

Command Type	Packets Sent	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
Write_Single_Coil	50	1799.167	26.183	2.138	(1794.942, 1803.391)
Read_Coils_Min	50	1799.267	26.458	2.160	(1794.998, 1803.535)
Read_Coils_Max_Best	50	1799.300	26.721	2.182	(1794.989, 1803.611)
Write_Multi_Coil_Min	50	1832.060	20.373	1.663	(1828.773, 1835.347)
Read_Coils_Max_Worst	50	3781.980	26.147	2.135	(3777.761, 3786.199)
Write_Multi_Coil_Max_Best	50	5427.767	100.128	8.175	(5411.612, 5443.921)
Write_Multi_Coil_Max_Worst	50	7853.620	75.425	6.158	(7841.451, 7865.789)

Table 5.4 details the results of the one-variable t-test performed on the invalid MODBUS/TCP commands that operate on holding registers in a system with

non-saturated Bloom filters. The table contains the number of packets sent, the mean CPU cycles, the standard deviation, the standard error of the mean, and the 95% confidence interval. The table is ordered by mean CPU cycles, equating to command processing time. The range of mean command processing time for commands that operate on coils is 1799.04 to 5490.093 cycles.

Table 5.4: Invalid Command Processing Times - H. Registers - Non-saturated Bloom Filter

Command Type	Packets Sent	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
Read_HR_Max_Best	50	1799.040	26.151	2.135	(1794.821, 1803.259)
Write_Single_HR	50	1799.093	26.823	2.190	(1794.766, 1803.421)
Read_HR_Min	50	1799.333	26.589	2.171	(1795.043, 1803.623)
Write_Multi_HR_Min	50	1866.780	38.051	3.107	(1860.641, 1872.919)
Read_HR_Max_Worst	50	1925.593	24.845	2.029	(1921.585, 1929.602)
Write_Multi_HR_Max_Best	50	5424.013	76.642	6.258	(5411.648, 5436.379)
Write_Multi_HR_Max_Worst	50	5490.093	46.230	3.775	(5482.635, 5497.552)

Table 5.5 details the results of the one-variable t-test performed on the invalid MODBUS/TCP commands that operate on coils in a system with saturated Bloom filters. The table contains the number of packets sent, the mean CPU cycles, the standard deviation, the standard error of the mean, and the 95% confidence interval. The table is ordered by mean CPU cycles, equating to command processing time. The range of mean command processing time for commands that operate on coils is 1880.207 to 96438.433 cycles.

Table 5.6 details the results of the one-variable t-test performed on the invalid MODBUS/TCP commands that operate on coils in a system with saturated Bloom filters.

Table 5.5: Invalid Command Processing Times - Coils - Saturated Bloom Filter

Command Type	Packets Sent	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
Write_Single_Coil	50	1880.207	20.339	1.661	(1876.925, 1883.488)
Read_Coils_Min	50	1921.727	55.202	4.507	(1912.820, 1930.633)
Write_Multi_Coil_Min	50	1970.847	28.393	2.318	(1966.266, 1975.428)
Read_Coils_Max_Best	50	3865.567	24.366	1.989	(3861.635, 3869.498)
Write_Multi_Coil_Max_Best	50	7921.587	79.199	6.467	(7908.809, 7934.365)
Read_Coils_Max_Worst	50	91828.733	48.199	3.935	(91820.957, 91836.510)
Write_Multi_Coil_Max_Worst	50	96438.433	119.637	9.768	(96419.131, 96457.736)

The table contains the number of packets sent, the mean CPU cycles, the standard deviation, the standard error of the mean, and the 95% confidence interval. The table is ordered by mean CPU cycles, equating to command processing time. The range of mean command processing time for commands that operate on coils is 1899.820 to 20698.633 cycles.

Table 5.6: Invalid Command Processing Times - H. Registers - Saturated Bloom Filter

Command Type	Packets Sent	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
Read_HR_Min	50	1899.820	25.918	2.116	(1895.638, 1904.002)
Write_Single_HR	50	1925.313	20.523	1.676	(1922.002, 1928.625)
Write_Multi_HR_Min	50	1977.827	40.836	3.334	(1971.238, 1984.415)
Read_HR_Max_Best	50	2055.700	55.924	4.566	(2046.677, 2064.723)
Write_Multi_HR_Max_Best	50	5653.993	114.679	9.364	(5635.491, 5672.496)
Read_HR_Max_Worst	50	7623.607	55.437	4.526	(7614.662, 7632.551)
Write_Multi_HR_Max_Worst	50	20698.633	58.975	4.815	(20689.118, 20708.148)

Much like the valid MODBUS/TCP workload, single operation commands have similar processing times and commands with more operations take longer. However, differences are apparent when comparing command processing times between the system with non-saturated Bloom filters and the system with saturated Bloom filters. In all cases, the non-saturated Bloom filter system performs better. A two sample t-test was performed on the differences between the two systems. Table 5.7 displays for each invalid command the mean CPU cycles for the non-saturated Bloom filter system, the saturated Bloom filter system, the difference between the mean CPU cycles of the systems, the 95% confidence interval for the difference between the means, and the p-value for the difference between the means. For every case, the p-value is extremely small, never larger than 5.899e-56. The range of differences is as low as 81.04 and as high as 88584.813 clock cycles equating to a time savings of 648.32 ns to 708.679 μs. The improvement of execution time for the non-saturated Bloom filter over the saturated Bloom filter ranges from 4.5% for the Write Single Coil command to 2328.06% for a maximum length Read Coils command with the invalid address in the worst location.

5.1.3 Non-MODBUS/TCP Workload.

Table 5.8 details the results of the one-variable t-test performed on the non-MODBUS/TCP workload for systems with non-saturated Bloom filters and saturated Bloom filters. The table contains the number of packets sent, the mean CPU cycles, the standard deviation, the standard error of the mean, and the 95% confidence interval. The table is ordered by mean CPU cycles, equating to command processing time. The range of mean command processing time for commands that operate on coils is 1337.507 to 1337.933 cycles.

Non-MODBUS/TCP command processing times are overall very low. It also appears that whether the Bloom filter is saturated or not has no effect on the processing time. This

Table 5.7: Bloom filter Saturation Command Processing Time Differences

Command Type	Mean CPU Cycles Non-saturated	Mean CPU Cycles Saturated	Difference Between Means	Confidence Interval of Difference Between Means (95%)	p-value
Write_Single_Coil	1799.167	1880.207	81.040	(75.711, 86.369)	2.246e-89
Read_HR_Min	1799.333	1899.820	100.487	(94.520, 106.453)	6.198e-102
Write_Multi_HR_Min	1866.780	1977.827	111.047	(102.078, 120.016)	9.233e-73
Read_Coils_Min	1799.267	1921.727	122.460	(112.608, 132.312)	5.005e-64
Write_Single_HR	1799.093	1925.313	126.220	(120.792, 131.648)	1.018e-131
Write_Multi_Coil_Min	1832.060	1970.847	138.787	(133.169, 144.404)	1.084e-135
Write_Multi_HR_Max_Best	5424.013	5653.993	229.980	(207.803, 252.157)	5.899e-56
Read_HR_Max_Best	1799.040	2055.700	256.660	(246.723, 266.597)	1.468e-120
Read_Coils_Max_Best	1799.300	3865.567	2066.267	(2060.456, 2072.078)	0.000e+00
Write_Multi_Coil_Max_Best	5427.767	7921.587	2493.820	(2473.302, 2514.338)	0.000e+00
Read_HR_Max_Worst	1925.593	7623.607	5698.013	(5688.234, 5707.792)	0.000e+00
Write_Multi_HR_Max_Worst	5490.093	20698.633	15208.540	(15196.496, 15220.584)	0.000e+00
Read_Coils_Max_Worst	3781.980	91828.733	88046.753	(88037.932, 88055.575)	0.000e+00
Write_Multi_Coil_Max_Worst	7853.620	96438.433	88584.813	(88562.071, 88607.556)	0.000e+00

Table 5.8: Non-MODBUS/TCP Processing Time

Command Type	Packets Sent	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
Non_Modbus_BF_Saturated	50	1337.507	21.882	1.787	(1333.976, 1341.037)
Non_Modbus_BF_Not_Saturated	50	1337.933	22.976	1.876	(1334.226, 1341.640)

makes sense as the commands are identified as not being MODBUS/TCP well before they would be dissected and processed through the Bloom filters.

5.1.4 Overall Analysis of Experiment 1.

Between workloads, similarities become apparent. Commands that perform multiple operations take longer than commands that perform one operation. Maximum length writes also take longer than their maximum length read counterparts. This is due to the additional processing that writes perform. The command that takes the longest time to process is the valid maximum length Write Multiple Coils command at 99550.633 clock cycles. In the 125 MHz system, this equates to 796.4 μs. Its invalid counterpart in the saturated Bloom filter system takes 96438.433 clock cycles or 771.5 μs. This difference is likely because the valid command is copied byte by byte for later usage whereas the invalid command is logged at a higher speed by copying four bytes at a time. In each of these cases, the processing time is under 1 ms, leaving a 4 ms leeway in order to process the relevant responses.

5.2 Results and Analysis of Experiment 2

This section presents the results of experiment 2. Results are presented in tabular format with notation the same as in Section 5.1.

5.2.1 Valid MODBUS/TCP Workload.

Table 5.9 details the results of the one-variable t-test performed on the valid MODBUS/TCP response workload for commands that operate on coils. The table contains the number of packets sent, the mean CPU cycles, the standard deviation, the standard error of the mean, and the 95% confidence interval. The table is ordered by mean CPU cycles, equating to response processing time. The range of mean command processing time for commands that operate on coils is 1549.320 to 294558.253 cycles.

Table 5.10 details the results of the one-variable t-test performed on the valid MODBUS/TCP response workload for commands that operate on holding registers. The table contains the number of packets sent, the mean CPU cycles, the standard deviation, the standard error of the mean, and the 95% confidence interval. The table is ordered by mean CPU cycles, equating to response processing time. The range of mean command processing time for commands that operate on coils is 1500.320 to 22332.380 cycles.

Table 5.9: Valid Response Processing Times - Coils

Command Type	Lines Monitored	Packets Received	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
Write_Single_Coil	1	50	1549.320	3.940	0.322	(1548.684, 1549.956)
Read_Coils_Min	1	50	1600.340	3.941	0.322	(1599.704, 1600.976)
Write_Multi_Coil_Min	1	50	1602.340	3.941	0.322	(1601.704, 1602.976)
Read_Coils_Max	1	50	287937.053	7.547	0.616	(287935.836, 287938.271)
Read_Coils_Max	5	50	289766.567	4.543	0.371	(289765.834, 289767.300)
Write_Multi_Coil_Max	1	50	292730.253	3.914	0.320	(292729.622, 292730.885)
Write_Multi_Coil_Max	5	50	294558.253	3.914	0.320	(294557.622, 294558.885)

Table 5.10: Valid Response Processing Times - Holding Registers

Command Type	Lines Monitored	Packets Received	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
Write_Single_HR	1	50	1500.320	3.883	0.317	(1499.694, 1500.946)
Write_Multi_HR_Min	1	50	1559.340	3.941	0.322	(1558.704, 1559.976)
Read_HR_Min	1	50	1562.340	3.941	0.322	(1561.704, 1562.976)
Write_Multi_HR_Max	1	50	19781.280	3.881	0.317	(19780.654, 19781.906)
Read_HR_Max	1	50	20560.380	4.053	0.331	(20559.726, 20561.034)
Write_Multi_HR_Max	5	50	21534.280	3.881	0.317	(21533.654, 21534.906)
Read_HR_Max	5	50	22332.380	4.053	0.331	(22331.726, 22333.034)

The minimal length responses all have very consistent processing times, ranging from 1500 to 1600 clock cycles. The consistency between them suggests that the similarity is due to the time it takes to query one ADC. The processing time responses for maximum length commands increases significantly over the processing time for minimum length commands. For responses to commands that affect coils it is interesting to note that again writes take longer to process than reads despite operating on 32 fewer addresses regardless of how many lines are monitored. This is contrary to responses to commands that affect holding registers where its clear that order is first decided by how many lines are monitored and then by command type.

5.2.2 Infected Response MODBUS/TCP Workload.

Table 5.11 details the results of the one-variable t-test performed on the infected response MODBUS/TCP workload for commands that operate on coils. The table contains the number of lines monitored, the number of packets sent, the mean CPU cycles, the standard deviation, the standard error of the mean, and the 95% confidence interval. The table is ordered by mean CPU cycles, equating to response processing time. The range of mean command processing time for commands that operate on coils is 3190.187 to 302845.960 cycles.

Table 5.11: Infected Response Processing Times - Coils

Command Type	Lines Monitored	Packets Received	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
Read_Coils_Min	1	50	3190.187	32.236	2.632	(3184.986, 3195.388)
Write_Multi_Coil_Min	1	50	3230.600	37.745	3.082	(3224.510, 3236.690)
Write_Single_Coil	1	50	3249.120	34.958	2.854	(3243.480, 3254.760)
Write_Multi_Coil_Max_Best	5	150	6257.527	37.535	3.065	(6251.471, 6263.583)
Write_Multi_Coil_Max_Best	1	150	6257.580	37.688	3.077	(6251.499, 6263.661)
Read_Coils_Max_Best	1	50	6953.767	41.009	3.348	(6947.150, 6960.383)
Read_Coils_Max_Best	5	50	6955.293	42.385	3.461	(6948.455, 6962.132)
Write_Multi_Coil_Max_Worst	1	50	297379.100	36.533	2.983	(297373.206, 297384.994)
Write_Multi_Coil_Max_Worst	5	50	299304.333	37.235	3.040	(299298.326, 299310.341)
Read_Coils_Max_Worst	1	50	302817.060	41.849	3.417	(302810.308, 302823.812)
Read_Coils_Max_Worst	5	50	302845.960	42.680	3.485	(302839.074, 302852.846)

Table 5.12 details the results of the one-variable t-test performed on the infected response MODBUS/TCP workload for commands that operate on holding registers. The table contains the number of lines monitored, the number of packets sent, the mean CPU cycles, the standard deviation, the standard error of the mean, and the 95% confidence

interval. The table is ordered by mean CPU cycles, equating to response processing time. The range of mean command processing time for commands that operate on coils is 2614.307 to 27213.313 cycles.

Table 5.12: Infected Response Processing Times - Holding Registers

Command Type	Lines Monitored	Packets Received	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
Write_Single_HR	1	50	2614.307	35.285	2.881	(2608.614, 2620.000)
Read_HR_Min	1	50	3150.913	36.991	3.020	(3144.945, 3156.881)
Write_Multi_HR_Min	1	50	3205.480	31.790	2.596	(3200.351, 3210.609)
Write_Multi_HR_Max_Best	1	50	6300.213	37.976	3.101	(6294.086, 6306.340)
Write_Multi_HR_Max_Best	5	50	6300.920	37.340	3.049	(6294.896, 6306.944)
Read_HR_Max_Best	1	50	6878.773	44.120	3.602	(6871.655, 6885.892)
Read_HR_Max_Best	5	50	6878.820	44.687	3.649	(6871.610, 6886.030)
Write_Multi_HR_Worst	1	50	24409.740	41.434	3.383	(24403.055, 24416.425)
Read_HR_Max_Worst	1	50	25347.687	45.355	3.703	(25340.369, 25355.004)
Write_Multi_HR_Max_Worst	5	50	26268.400	40.482	3.305	(26261.869, 26274.931)
Read_HR_Max_Worst	5	50	27213.313	46.432	3.791	(27205.822, 27220.805)

In the infected response MODBUS/TCP workload, minimum length commands again take about the same time to process, around 3200 clock cycles. The exception to this is the Write Single Holding Register command. This response's mean response processing time is 2614.307 clock cycles, about 600 clock cycles fewer than the other minimum length commands. The difference is not evident in the response processing code but could be due to an optimization in the compiled code.

For maximum length commands, some interesting cases to note are responses where the location of the infected address are best. For these cases, there appears to be no or very little difference between cases of one monitored line versus five monitored lines. This is because the algorithm found that the first address checked reported an incorrect line value, causing the algorithm to immediately stop processing and log the packet. This is entirely intended behavior. Furthermore, responses to commands affecting holding registers again show an ordering where more lines monitored have a stronger effect on whether a command takes longer. For responses to commands affecting coils read commands appear seem to take longer than write commands. This indicates that for holding registers ADC query time dominates whereas in coils number of operations dominates.

5.2.3 Overall Analysis of Experiment 2.

The overall range of response processing for valid MODBUS/TCP responses was 1500.32 to 294558.253 clock cycles whereas for infected MODBUS/TCP responses the range was 2614.307 to 302845.96 clock cycles. The infected response MODBUS/TCP range is higher for both the low and high range, demonstrating the cost of logging packets. For maximum length coil commands, it was noted for the uninfected workload that write commands take longer than read commands. This trend is reversed for infected responses and is likely due to an interaction between the infected factor and total number of operations.

5.3 Results and Analysis of Experiment 3

Experiment 3 required data processing from the results Experiments 1 and 2 before execution could begin. In order to determine the exact command frequencies for Valid Operation Break and Invalid Command - Saturated Break the total processing time of the slowest valid command needed to be calculated and the slowest invalid command for the saturated Bloom filter system needed to be determined. Table 5.13 shows each valid command, its mean command processing time, the number of monitored lines for its response, the mean response processing time for its response with the corresponding number of monitored lines, and the total processing time. The table is ordered by total command processing time from smallest to largest.

As can be seen, the maximum length Write Multiple Coils command with five monitored lines has the highest total processing time. This equates to 3.15 ms per command. Inverting this result provided the fastest the system can run in the worst case, yielding 317 commands per second. Since the required command frequency for the experiment was meant to break valid operation, 317 commands per second was too slow. 350 commands per second was chosen instead.

Table 5.13: Valid Workload Total Processing Time

Command Type	Mean CPU Cycles (Command)	Number of Lines Monitored	Mean CPU Cycles (Response)	Total Processing Time
Write_Single_Coil	2528.893	1	1549.32	4078.213
Write_Single_HR	2582.213	1	1500.32	4082.533
Read_HR_Min	2556.947	1	1562.34	4119.287
Read_Coils_Min	2551.307	1	1600.34	4151.647
Write_Multi_Coil_Min	2629.66	1	1602.34	4232
Write_Multi_HR_Min	2726.927	1	1559.34	4286.267
Read_HR_Max	8271.887	1	20560.38	28832.27
		5	22332.38	30604.27
Write_Multi_HR_Max	23897.11	1	19781.28	43678.39
		5	21534.28	45431.39
Read_Coils_Max	92509.95	1	287937.1	380447
		5	289766.6	382276.5
Write_Multi_Coil_Max	99550.63	1	292730.3	392280.9
		5	294558.3	394108.9

Selecting Invalid Command - Saturated Break was easier since total processing time was not necessary, only the command processing time was needed. In this case, the slowest command processing time for the saturated Bloom filter system is Write Multiple Coils with the invalid address being in the worst position. The mean CPU cycles for this command is 96438.433 cycles, equating to 771.82 μs per command. Inverting this provided a maximum command frequency of 1295 commands per second. Since breaking the operation was necessary, 1400 commands per second was chosen for Invalid Command Saturate Break.

5.3.1 Valid MODBUS/TCP Results.

Table 5.14 details the results of the one-variable t-tests performed on the packets logged and packets inspected for the worst case valid MODBUS/TCP command, maximum length Write Multiple Coils. The table shows the Bloom filter false positive rate, the command frequency, the number of packets sent, the mean number of packets logged, the mean number of packets inspected, the 95% confidence interval for the number of packets logged, and the 95% confidence interval for the number of packets inspected. The table is ordered by Bloom filter false positive rate then by command frequency. Figure 5.1 graphs the mean number of packets logged and the mean number of packets intercepted for the Bloom filter false positive rates of saturated and non-saturated shown in Table 5.14.

Table 5.14: Valid MODBUS/TCP Workload

Bloom filter False Positive Rate	Command Frequency	Packets Sent	Mean Packets Logged	Mean Packets Inspected	Confidence Int. Packets Logged (95%)	Confidence Int. Packets Inspected (95%)
Non-saturated	200	10000	0.000	10000.000	(0.000, 0.000)	(10000.000, 10000.000)
	350	10000	448.000	9776.000	(448.000, 448.000)	(9776.000, 9776.000)
	1400	10000	2366.667	4493.333	(2365.478, 2367.856)	(4492.360, 4494.307)
	5000	10000	1164.000	1851.333	(1163.369, 1164.631)	(1850.792, 1851.874)
Saturated	200	10000	0.000	10000.000	(0.000, 0.000)	(10000.000, 10000.000)
	350	10000	449.000	9776.000	(444.697, 453.303)	(9776.000, 9776.000)
	1400	10000	2344.667	4475.667	(2281.123, 2408.211)	(4355.852, 4595.482)
	5000	10000	1122.000	1812.667	(1057.556, 1186.444)	(1735.126, 1890.207)

At 200 commands per second the SIEVE system operates correctly; that is, all 10000 packets are inspected and zero are logged. At 350 commands per second and subsequent

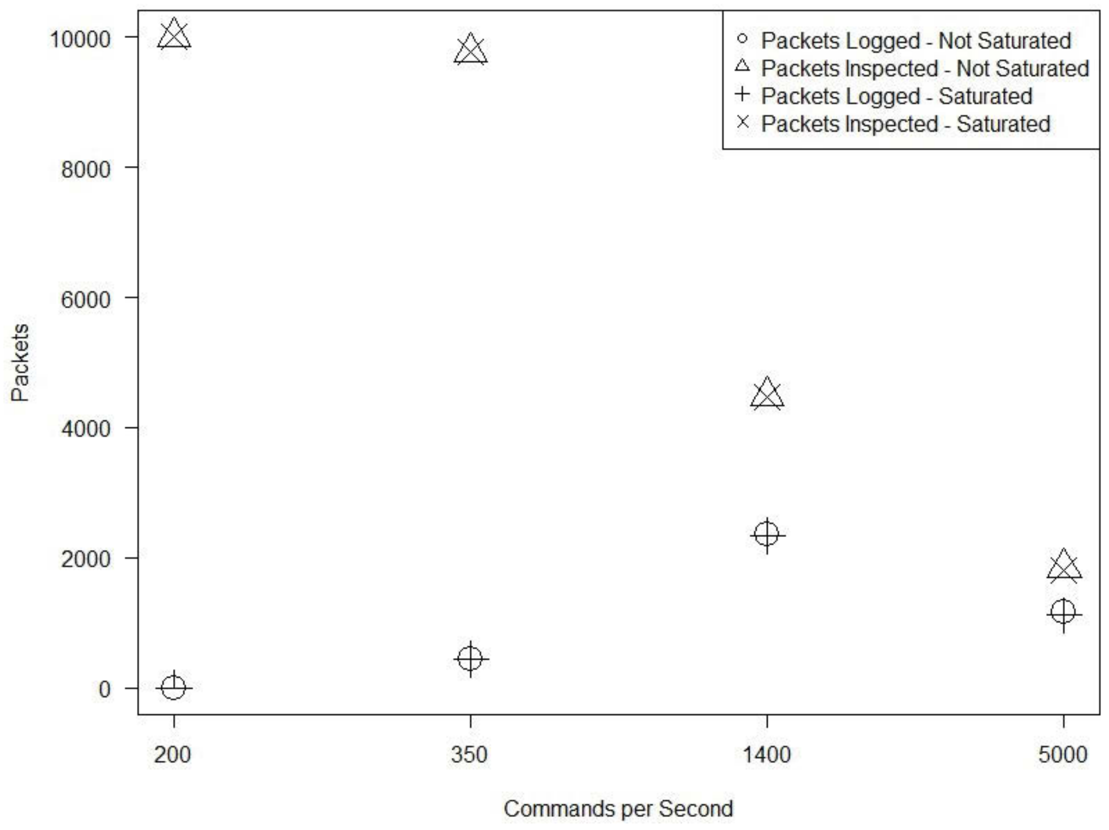


Figure 5.1: Valid MODBUS/TCP Workload Packet Intercepts

command frequencies where valid operation begins to break down and false positives are reported. This is due to the system not being able to process packets fast enough such that the receive buffers of the Ethernet core fill up, causing packets to be dropped. The dropped packets cause the algorithm to behave erratically because the system will either see commands without responses or responses that do not correspond to the previous command. An interesting feature, visible in Figure 5.1, is that the packets logged at 5000 commands per second is fewer than the packets logged at 1400 commands per second. This is explained by looking at the number of packets inspected for both points. At 1400 commands per second, approximately 4490 packets are inspected while about 2350 are logged. At 5000 commands per second approximately 1850 packets are inspected while

1160 are logged. The proportion of logged packets to inspected packets is greater for the 5000 commands per second case indicating a higher false positive rate.

Additionally, the non-saturated bloom filter system inspected more packets on average than the saturated bloom filter system but logged approximately the same proportion.

5.3.2 Invalid Command MODBUS/TCP Workload.

Table 5.15 details the results of the one-variable t-tests performed on the packets logged and packets inspected for the worst case invalid command MODBUS/TCP workload, maximum length Write Multiple Coils with an invalid address as the last address operated on. The table shows the Bloom filter false positive rate, the command frequency, the number of packets sent, the mean number of packets logged, the mean number of packets inspected, the 95% confidence interval for the number of packets logged, and the 95% confidence interval for the number of packets inspected. The table is ordered by Bloom filter false positive rate then by command frequency. Figure 5.2 graphs the mean number of packets logged and the mean number of packets intercepted for the Bloom filter false positive rates of saturated and non-saturated shown in Table 5.15.

Table 5.15: Invalid MODBUS/TCP Command Workload

Bloom filter False Positive Rate	Command Frequency	Packets Sent	Mean Packets Logged	Mean Packets Inspected	Confidence Int. Packets Logged (95%)	Confidence Int. Packets Inspected (95%)
Not Saturated	200	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
	350	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
	1400	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
	5000	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
Saturated	200	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
	350	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
	1400	10000	9382.333	9382.333	(9378.539, 9386.128)	(9378.539, 9386.128)
	5000	10000	3896.667	3896.667	(3726.502, 4066.832)	(3726.502, 4066.832)

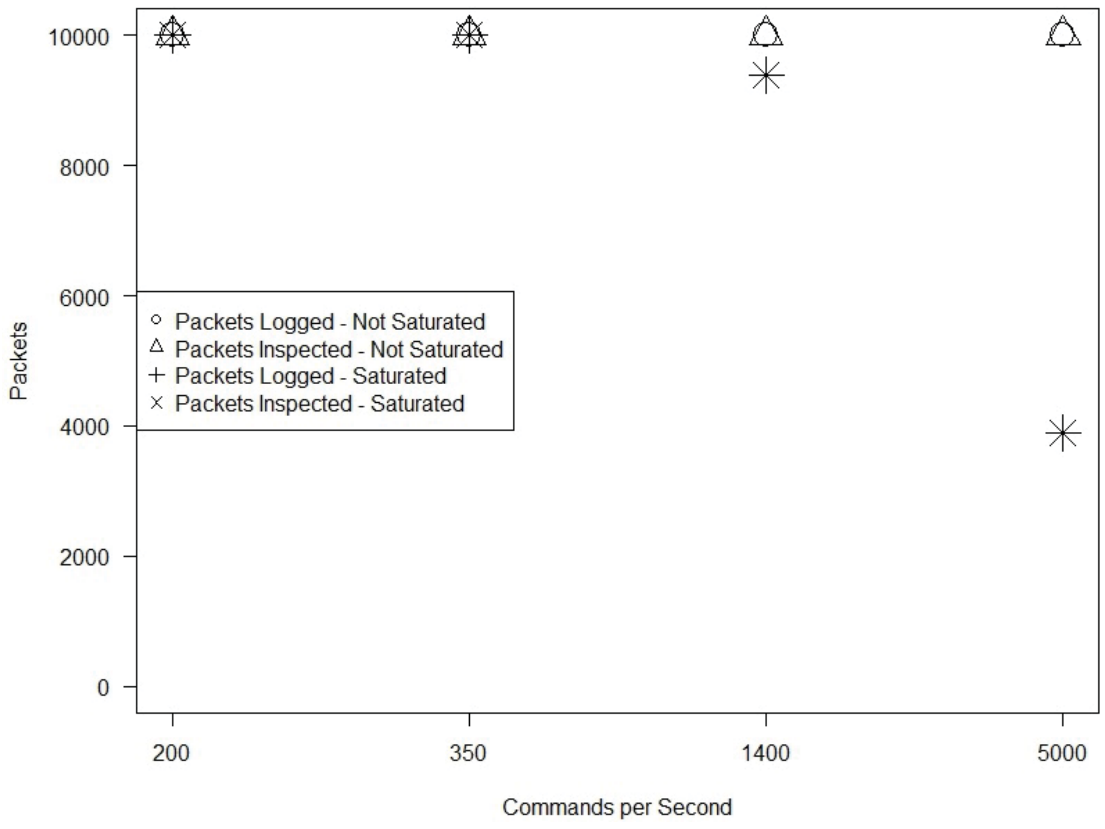


Figure 5.2: Invalid Command MODBUS/TCP Packet Intercepts

As can be seen in Figure 5.2, logged and inspected packets begin dropping at 1400 commands per second. This is expected since 1400 commands per second was supposed to degrade operations, distinguishing it from the non-saturated Bloom filter system. In this case, at both 1400 commands per second and 5000 commands per second, the non-saturated Bloom filter inspects and logs all packets. The saturated Bloom filter inspects fewer than 3900 packets at the 5000 commands per second level, dropping over half of the packets sent.

On logging ability, both systems logged every packet they received, meaning there are zero false negatives.

5.3.3 Infected Response MODBUS/TCP Workload.

Table 5.16 details the results of the one-variable t-tests performed on the packets logged and packets inspected for the worst case infected response MODBUS/TCP workload, maximum length Write Multiple Coils with five monitored lines and the last monitored address being infected. The table shows the Bloom filter false positive rate, the command frequency, the number of packets sent, the mean number of packets logged, the mean number of packets inspected, the 95% confidence interval for the number of packets logged, and the 95% confidence interval for the number of packets inspected. The table is ordered by Bloom filter false positive rate then by command frequency. Figure 5.3 graphs the mean number of packets logged and the mean number of packets intercepted for the Bloom filter false positive rates of saturated and non-saturated shown in Table 5.16.

Table 5.16: Infected MODBUS/TCP Response Workload

Bloom filter False Positive Rate	Command Frequency	Packets Sent	Mean Packets Logged	Mean Packets Inspected	Confidence Int. Packets Logged (95%)	Confidence Int. Packets Inspected (95%)
Not Saturated	200	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
	350	10000	9732.000	9732.000	(9732.000, 9732.000)	(9732.000, 9732.000)
	1400	10000	4342.667	4343.000	(4342.568, 4342.766)	(4342.896, 4343.104)
	5000	10000	1820.333	1820.333	(1820.181, 1820.485)	(1820.181, 1820.485)
Saturated	200	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
	350	10000	9732.000	9732.000	(9732.000, 9732.000)	(9732.000, 9732.000)
	1400	10000	4326.667	4326.667	(4200.872, 4452.462)	(4200.872, 4452.462)
	5000	10000	1806.667	1806.667	(1752.166, 1861.167)	(1752.166, 1861.167)

For the infected response MODBUS/TCP workload both systems' performance begins to drop at 350 commands per second. At 1400 commands per second, fewer than half of the packets sent are inspected and logged. This is expected considering that the processing times for valid responses are slightly faster than processing times for infected

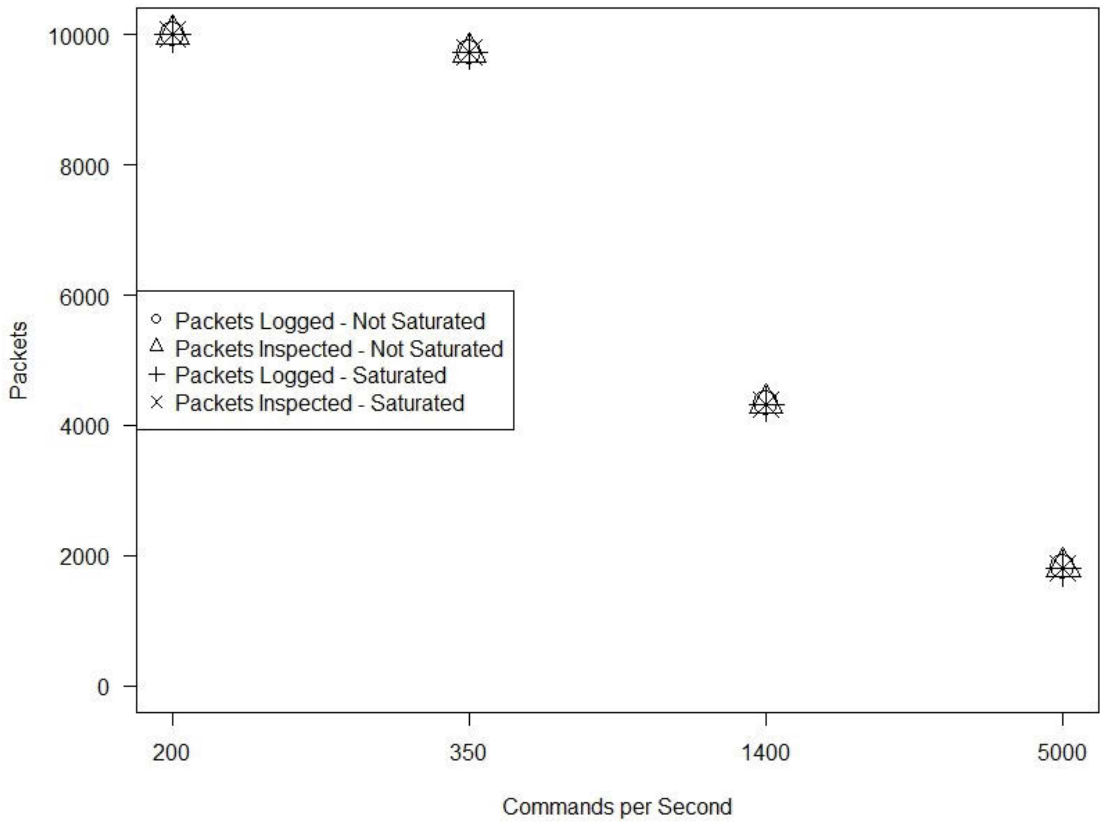


Figure 5.3: Infected Response Processing Time - Holding Registers

responses. However, at the goal rate of 200 commands per second, all packets sent are logged and inspected.

5.3.4 Non-MODBUS/TCP Workload.

Table 5.17 details the results of the one-variable t-tests performed on the packets logged and packets inspected for the non-MODBUS/TCP workload. The table shows the Bloom filter false positive rate, the command frequency, the number of packets sent, the mean number of packets logged, the mean number of packets inspected, the 95% confidence interval for the number of packets logged, and the 95% confidence interval for the number of packets inspected. The table is ordered by Bloom filter false positive rate then by command frequency. Figure 5.4 graphs the mean number of packets logged and the mean number of packets intercepted for the Bloom filter false positive rates of saturated and non-saturated shown in Table 5.17.

Table 5.17: Non-MODBUS/TCP Workload

Bloom filter False Positive Rate	Command Frequency	Packets Sent	Mean Packets Logged	Mean Packets Inspected	Confidence Int. Packets Logged (95%)	Confidence Int. Packets Inspected (95%)
Not Saturated	200	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
	350	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
	1400	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
	5000	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
Saturated	200	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
	350	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
	1400	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)
	5000	10000	10000.000	10000.000	(10000.000, 10000.000)	(10000.000, 10000.000)

For the non-MODBUS/TCP workload, all of the packets sent are logged and inspected at all of the command frequencies tested. The Bloom filter false positive rate did not have any effect on packet logging or inspection.

5.3.5 Overall Analysis of Experiment 3.

Among the workloads tested, it is clear that invalid MODBUS/TCP commands and non-MODBUS/TCP traffic are handled more quickly by the SIEVE system. This is likely because they are identified as being invalid, negating the need to process a response, or are identified as not being MODUS/TCP in hardware. Valid MODBUS/TCP traffic and the infected MODBUS/TCP response both inspected about the same number of packets with all of the invalid responses correctly being logged. Since the workloads used for this test are all worst case and all tests showed 100% inspection rates at the 200 command per second frequency any other workload should be able to be handled at 200 commands per

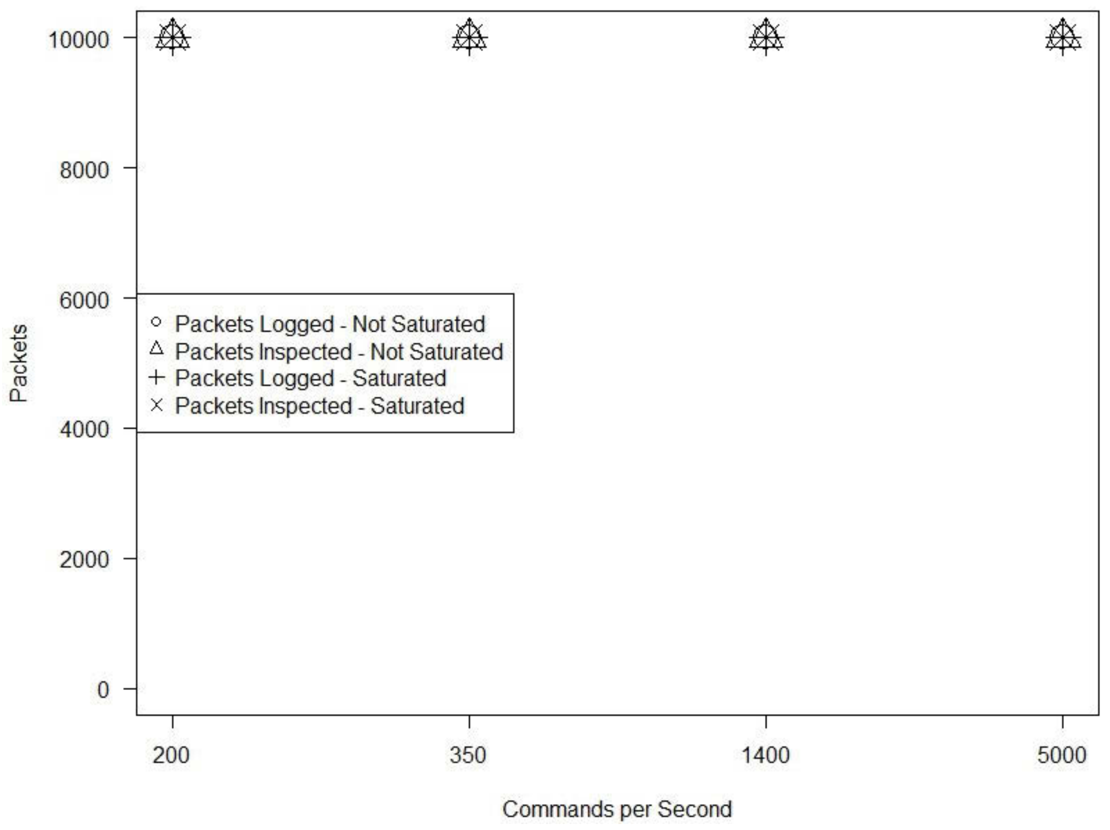


Figure 5.4: Invalid Response Processing Time - Holding Registers

second or slower. This indicates that the goal of the research is achieved since any command with a response can be processed in under 5 ms.

The Bloom filter false positive rate only showed significant improvements for the invalid MODBUS/TCP command workload. This is expected since only invalid commands should get rejected by the MODBUS inspection core.

VI. Conclusion

This chapter summarizes the goals and conclusions of the research. Section 6.1 summarizes the results of the experiments and whether the goals are met. Section 6.2 explains the significance of the research. Lastly, Section 6.3 describes further work that can be done with this research.

6.1 Conclusions

The goal of this research is to determine if a specifications-based SCADA integrity verification system could correctly verify the integrity of incoming network traffic and device response in realtime. Three experiments are performed to demonstrate this. The first experiment shows that in the worst case, commands take no longer than 796.4 μs for processing. The second experiment shows that in the worst case 3.15 ms is required to process a response. These results feed into the third experiment which shows that for all worst case workload types 100% of packets are inspected and, if necessary, logged at a 200 command per second frequency. This means that no matter the packet type, it is inspected/logged in under five ms, meeting the research goal.

For the secondary goal of the experiment, Experiment 1 shows the benefit of a properly tuned Bloom filter. For the worst case invalid MODBUS/TCP command, a Write Multiple Coils command that operated on 1968 consecutive coils, the last of which is an invalid address, the non-saturated Bloom filter system saves 88584.813 clock cycles over the saturated Bloom filter system. Equating that to real time, the non-saturated Bloom filter system saves 708.7 μs or an improvement of 1127.95%. This also translates into improved performance in Experiment 3 where at a 5000 command per second frequency the non-saturated Bloom filter system logs all packets sent whereas the saturated Bloom filter system logs less than half. Overall, this demonstrates the value of the Bloom filter in a SCADA integrity verification system.

6.2 Significance

The demand for improved access to and setup of SCADA networks has ultimately left them vulnerable to attack. Current research efforts focus more on detection of attacks or malicious traffic. Detecting a compromised device is more difficult as it requires knowledge of what the PLC is actually doing. Together, these services represent the verification of the integrity of the inputs and outputs of the system. This research produced the first SCADA integrity verification device.

As the first of its kind, it can be difficult to express the significance of the SIEVE system. It can be used as a high-efficiency intrusion detection system for PLCs in addition to the more significant role it can take on as a forensic device. As stated above, detecting a compromised device is more difficult due to requiring knowledge of exactly how the PLC is operating. Since the SIEVE system is specification based, it knows what the PLC should be doing and checks it as commands are sent. This means that a Stuxnet-like attack can be defeated as the operators will be warned when the PLC is not doing what it is instructed to.

6.3 Future Work

The first, and perhaps foremost, area for future work with this research would be to extending its support for 16-bit registers. This is important as MODBUS supports 16-bit registers; however, due to limitations of the SIEVE system this was not possible to implement for this research. The limiting factor for the SIEVE system was the ADC used for current sensing. The 12-bit ADS7818p, due to the limitations discussed in Chapter 3, can truly only distinguish eight-bits for this kind of application. Sixteen-bit support will require at minimum a 20-bit ADC for the same reasons.

The second area for future work would also be improving the current sense circuit. The current implementation appears to have grounding issues, meaning that distinguishing voltage levels is difficult for the ADC. This could be fixed through use of a printed circuit board specially designed for the application. Another improvement could be the current sense resistor. At 200 Ω it dissipates 80 mW of power which could affect the performance of the PLC. A smaller resistor would lower this power dissipation but likely require an amplifier stage in order to boost the voltage drop into a range that can be sensed by the ADC.

Another area for future work with this research would be in improving response processing time. As it is implemented currently, processing a response can take a significant amount of time, up to four times the amount of time needed to process a command in some cases. One way which this can be accomplished is through optimization of the process itself or taking advantage of the second PowerPC core on the FPGA and dividing the work.

Lastly, the warnings sent to operators is currently only visible on a single computer screen. Better warning mechanisms could be investigated such as sending an e-mail or an SMS text message. Since the SIEVE system operates in realtime, its warning capabilities could also be integrated into other systems. This could allow more automatic actions to be taken.

Appendix A: Experimental Scenarios

This appendix contains listings of the scenarios used in Experiments 1, 2, and 3. Section A.1 contains the scenarios for Experiment 1, Section A.2 contains the scenarios for Experiment 2, and Section A.3 contains the scenarios for Experiment 3.

A.1 Experiment 1 Scenarios

Table A.1: Experiment 1 Scenarios

Workload	Command Type	Command Length	Invalid Location	Bloom Filter False Positive Rate
Valid MODBUS/TCP	Read Coils	Min		1
Valid MODBUS/TCP	Read Coils	Max		1
Valid MODBUS/TCP	Read H. Registers	Min		1
Valid MODBUS/TCP	Read H. Registers	Max		1
Valid MODBUS/TCP	Write Multiple Coils	Min		1
Valid MODBUS/TCP	Write Multiple Coils	Max		1
Valid MODBUS/TCP	Write Multiple H. Registers	Min		1
Valid MODBUS/TCP	Write Multiple H. Registers	Max		1
Valid MODBUS/TCP	Write Single Coil			1
Valid MODBUS/TCP	Write Single H. Reg			1
Invalid MODBUS/TCP	Read Coils	Min		1
Invalid MODBUS/TCP	Read Coils	Max	Best	1
Invalid MODBUS/TCP	Read Coils	Max	Worst	1
Invalid MODBUS/TCP	Read H. Registers	Min		1
Invalid MODBUS/TCP	Read H. Registers	Max	Best	1
Invalid MODBUS/TCP	Read H. Registers	Max	Worst	1
Invalid MODBUS/TCP	Write Multiple Coils	Min		1
Invalid MODBUS/TCP	Write Multiple Coils	Max	Best	1
Invalid MODBUS/TCP	Write Multiple Coils	Max	Worst	1
Invalid MODBUS/TCP	Write Multiple H. Registers	Min		1
Invalid MODBUS/TCP	Write Multiple H. Registers	Max	Best	1
Invalid MODBUS/TCP	Write Multiple H. Registers	Max	Worst	1
Invalid MODBUS/TCP	Write Single Coil			1
Invalid MODBUS/TCP	Write Single H. Reg			1

Workload	Command Type	Command Length	Invalid Location	Bloom Filter False Positive Rate
Invalid MODBUS/TCP	Read Coils	Min		Not 1
Invalid MODBUS/TCP	Read Coils	Max	Best	Not 1
Invalid MODBUS/TCP	Read Coils	Max	Worst	Not 1
Invalid MODBUS/TCP	Read H. Registers	Min		Not 1
Invalid MODBUS/TCP	Read H. Registers	Max	Best	Not 1
Invalid MODBUS/TCP	Read H. Registers	Max	Worst	Not 1
Invalid MODBUS/TCP	Write Multiple Coils	Min		Not 1
Invalid MODBUS/TCP	Write Multiple Coils	Max	Best	Not 1
Invalid MODBUS/TCP	Write Multiple Coils	Max	Worst	Not 1
Invalid MODBUS/TCP	Write Multiple H. Registers	Min		Not 1
Invalid MODBUS/TCP	Write Multiple H. Registers	Max	Best	Not 1
Invalid MODBUS/TCP	Write Multiple H. Registers	Max	Worst	Not 1
Invalid MODBUS/TCP	Write Single Coil			Not 1
Invalid MODBUS/TCP	Write Single Reg			Not 1
Not Modbus				1
Not Modbus				Not 1

A.2 Experiment 2 Scenarios

Table A.2: Experiment 2 Scenarios

Infection	Response	Length	Inv Loc	Monitored
Not Infected	Read Coils	Min		1
Not Infected	Read Coils	Max		1
Not Infected	Read Coils	Max		5
Not Infected	Read H. Registers	Min		1
Not Infected	Read H. Registers	Max		1
Not Infected	Read H. Registers	Max		5
Not Infected	Write Multiple Coils	Min		1
Not Infected	Write Multiple Coils	Max		1
Not Infected	Write Multiple Coils	Max		5

Infection	Response	Length	Inv Loc	Monitored
Not Infected	Write Multiple H. Registers	Min		1
Not Infected	Write Multiple H. Registers	Max		1
Not Infected	Write Multiple H. Registers	Max		5
Not Infected	Write Single Coil			1
Not Infected	Write Single H. Register			1
Infected	Read Coils	Min		1
Infected	Read Coils	Max	Best	1
Infected	Read Coils	Max	Worst	1
Infected	Read Coils	Max	Best	5
Infected	Read Coils	Max	Worst	5
Infected	Read H. Registers	Min		1
Infected	Read H. Registers	Max	Best	1
Infected	Read H. Registers	Max	Worst	1
Infected	Read H. Registers	Max	Best	5
Infected	Read H. Registers	Max	Worst	5
Infected	Write Multiple Coils	Min		1
Infected	Write Multiple Coils	Max	Best	1
Infected	Write Multiple Coils	Max	Worst	1
Infected	Write Multiple Coils	Max	Best	5
Infected	Write Multiple Coils	Max	Worst	5
Infected	Write Multiple H. Registers	Min		1
Infected	Write Multiple H. Registers	Max	Best	1
Infected	Write Multiple H. Registers	Max	Worst	1
Infected	Write Multiple H. Registers	Max	Best	5
Infected	Write Multiple H. Registers	Max	Worst	5
Infected	Write Single Coil			1
Infected	Write Single H. Register			1

A.3 Experiment 3 Scenarios

Table A.3: Experiment 3 Scenarios

Workload	PLC Infection	Command Frequency	BF False Positive Rate
Valid MODBUS/TCP	Uninfected	200	Not 1
Valid MODBUS/TCP	Uninfected	350	Not 1
Valid MODBUS/TCP	Uninfected	1400	Not 1
Valid MODBUS/TCP	Uninfected	5000	Not 1
Valid MODBUS/TCP	Uninfected	200	1
Valid MODBUS/TCP	Uninfected	350	1
Valid MODBUS/TCP	Uninfected	1400	1
Valid MODBUS/TCP	Uninfected	5000	1
Valid MODBUS/TCP	Infected	200	Not 1
Valid MODBUS/TCP	Infected	350	Not 1
Valid MODBUS/TCP	Infected	1400	Not 1
Valid MODBUS/TCP	Infected	5000	Not 1
Valid MODBUS/TCP	Infected	200	1
Valid MODBUS/TCP	Infected	350	1
Valid MODBUS/TCP	Infected	1400	1
Valid MODBUS/TCP	Infected	5000	1
Invalid MODBUS/TCP Command		200	Not 1
Invalid MODBUS/TCP Command		350	Not 1
Invalid MODBUS/TCP Command		1400	Not 1
Invalid MODBUS/TCP Command		5000	Not 1
Invalid MODBUS/TCP Command		200	1
Invalid MODBUS/TCP Command		350	1
Invalid MODBUS/TCP Command		1400	1
Invalid MODBUS/TCP Command		5000	1
Non-MODBUS/TCP		200	Not 1
Non-MODBUS/TCP		350	Not 1
Non-MODBUS/TCP		1400	Not 1
Non-MODBUS/TCP		5000	Not 1
Non-MODBUS/TCP		200	1
Non-MODBUS/TCP		350	1
Non-MODBUS/TCP		1400	1
Non-MODBUS/TCP		5000	1

Appendix B: Emulated PLC Design

This appendix covers the operation and design of the emulated PLC. Section B.1 explains the basic operation of device, and Section B.2 covers the design of the Digital to Analog Converter (DAC) Bank.

B.1 Operation

The function of the emulated PLC is to behave like a PLC. This means it needs to respond to sent commands and update its connected lines if necessary. Since the workloads are known a priori, the responses sent are not calculated and built but read from a pcap file stored on a compact flash card. This eliminates complexity in the code at the cost of flexibility as the pcap file must be updated for each workload. Updating the analog lines is accomplished using DAC current sources. Their operation is described below in Section B.2.

B.2 DAC Bank

The DAC bank serves as a single point where the CPU can interact with the DAC modems. It also provides the serial clock from which the DACs clock their data. This top level entity contains five DACs, each set to a different 18-bit address. The DAC selected for this design is the AD5410 from Analog Devices. The AD5410 was chosen due to being a Digital to Analog Current Source. The advantage of a straight conversion to current rather than sourcing a voltage first is that it results in a much simpler circuit. In fact, the output of the AD5410 can be used to directly drive a 4 mA to 20 mA current loop, allowing it to be used as the current source in Figure 3.10.

The AD5410 is a 12-bit digital to analog current source. It's output current, based on the 12-bit input, is configurable to three ranges: 0 mA to 20 mA, 0 mA to 24 mA, and 4 mA to 20 mA. Since this research focuses on 4 mA to 20 mA control, the third range is chosen for operation. The transfer function for the AD5410 is

$$I_{out} = \left(\frac{16\ mA}{2^{12}}\right) * D + 4\ mA \tag{B.1}$$

where D is the 12-bit data value [Dev13]. The chip is capable of updating its current from 4 mA to 20 mA in 6 μs.

Data written to the DAC is sent three bytes at a time. The first byte is a register address while the latter two bytes are data. The AD5410 has five registers: NOP register, data register, readback register, control register, and reset register. For this research only the data and control registers are necessary. Figure B.1 shows the write timing to the DAC. The three bytes are output one bit at a time, from the most significant bit to the least significant, over 24 clock cycles. The data is latched into a shift register on the chip when the latch signal transitions from low to high. At the start of operation, 0x551005 is written to the chip. These three bytes write to the control register, select the 4 mA to 20 mA range, and enable output. Subsequent writes are to the data register and will alter the output current.

Like the ADC Bank, the DAC bank also generates the serial clock for the chips. This is done in the same way, using a counter to toggle the new, slower clock up or down. For the AD5410s this clock rate is 10.42 MHz. The data is directed to the correct DAC modem by matching an address that is hardcoded in VHDL to one that is written to the core by the CPU. If the address is a match, the selected modem outputs the data according to the timing shown above.

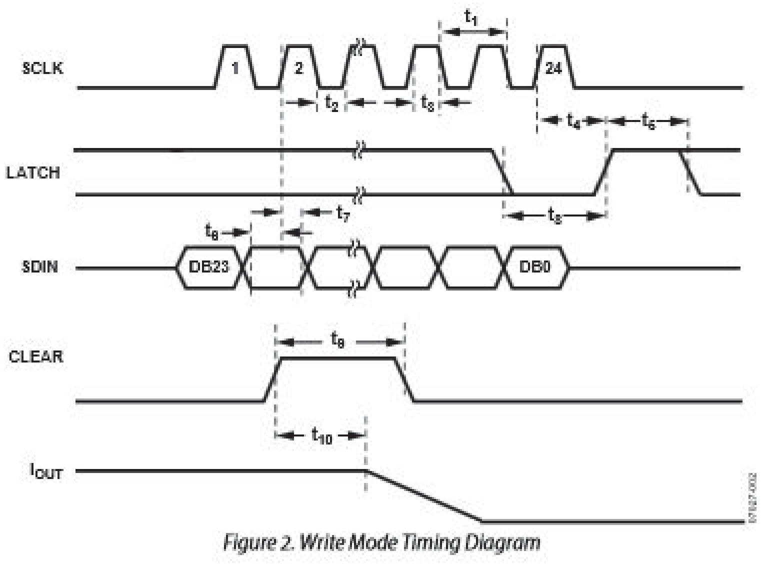


Figure B.1: DAC Write Timing [Dev13]

Appendix C: SIEVE System Setup

This appendix covers the configuration and setup of the SIEVE system. Section C.1 is a hardware description of the system, Section C.2 explains how to configure the Bloom filters and prepare them for loading to the FPGA, and Section C.3 shows how to load the system onto the ML507 development board using the Base System Builder in the Xilinx Platform Studio.

C.1 Hardware Description

This section describes each component of the SIEVE system and the role it takes in overall operation.

C.1.1 Processor.

The SIEVE system uses one of the embedded PowerPC 440 cores. This CPU executes the software application.

C.1.2 Processor Local Bus.

The Processor Local Bus connects the CPU to other peripherals on the development board. Communication with different Intellectual Property (IP) cores is accomplished using software registers and a 32-bit address system.

C.1.3 Block Random Access Memory.

The Block RAM structure is configured to have 128 KB of memory. It stores the main program as well as the stack and the heap.

C.1.4 XPS Tri-mode Ethernet Media Access Controller.

This IP core enables communication with the onboard Ethernet PHY chip. It is configured to be in promiscuous mode, receiving Ethernet frames regardless of their destination address. It is also configured to operate in GMII mode, enabling gigabit network speeds.

C.1.5 XPS Direct Memory Access Controller.

The XPS DMA controller is used to enable fixed length burst transfers up to 64 bytes long across the PLB. This speeds up data transfer from one IP core to another through elimination of overhead cycles used in negotiating single beat transfers.

C.1.6 Static Random Access Memory.

The ML507 has one MB of off chip SRAM. This memory is used for storing the function code valid for each address and the valid data ranges for the holding registers. The function code check is stored as one byte per address and represents the entire MODBUS memory space, requiring 256 KB of memory. The range check stores the low value and high value for each MODBUS holding register, storing each in two bytes (four bytes total). Combining both the low and high range check, another 256 KB of memory is required. In total, half of this module is used for storage.

C.1.7 Synchronous Dynamic Random Access Memory.

The ML507 has one off chip 256 MB SDRAM module and is used for storing the packet log file. The SRAM memory is not used for this since the log file can grow indefinitely. Since only 512 KB of memory was left in the SRAM, it was decided to store the file in the SDRAM where it can grow for longer.

C.1.8 XPS System ACE Controller.

This component allows reading and writing of files to compact flash cards. On bootup, the function code check and range checks are loaded into the SRAM from this module. When operation is complete or the log file needs to be inspected, the log file is copied to the card using this module.

C.1.9 XPS Universal Asynchronous Receiver/Transmitter Lite.

This IP core controls the RS232 connection using a UART. It is configured to communicate at a 115200 baud rate and is used to communicate device status and warnings to the operators.

C.1.10 Custom Timer.

This IP core was written to provide the number of microseconds and seconds since it was last reset. Both are used when timestamping a logged packet. The microseconds are determined by maintaining a counter that counts up once every system clock cycle. When 125 clock cycles have passed, the microsecond counter is incremented by one. After the microsecond counter has been incremented 1e6 times, the seconds counter is incremented. Both the seconds counter and microseconds counter can be reset by the CPU.

C.1.11 MODBUS Inspection Core.

The design and operation of this core is covered in Section 3.3.1.

C.1.12 ADC Bank.

The design and operation of this core is covered in Section 3.3.3.

C.2 Bloom Filter Configuration

This section covers the configuration of the Bloom filters and preparing them to be loaded onto the ML507 development board.

1. To begin, copy the following files into a directory that Matlab can operate in:

    - plc.txt
    - mpack_creator.m
    - bf_gen_k_10_m_16k_read.m
    - bf_gen_k_10_m_16k.m

2. Next, set up plc.txt to reflect the plc. The format is as folows:

    ```
    <type> <address> <low> <high> <increment>  <function_codes>
    ```

    where <type> is the MODBUS construct type (c=coil, d=discrete input, h= holding register, i=input register), <address> is the MODBUS address, <low> is the low value a register should take, <high> is the high value, and <increment> is how the data values should increase from low to high. The last field, <function_codes>, is an encoding of how the construct can be accessed. It is a four character long string and informs the mpack_creator.m script as to which function codes are valid for that construct. The string is set up as follows:

```
<read single><read multiple><write single><write multiple>
```

An 's' in the <read single> field or 'm' in the <read multiple> fields signify that the the construct can be read. An 's' in the <write single> field means it can be written to using a Write_Single command and an 'm' in the <write multiple> field means it can be written to using a Write_Multiple command. An 'x' should be used when a function code should not be used. Some examples are:

```
d 55 0 0 0 smxx
h 23 20 150 5 smsx
```

The discrete input is at address 55, which gets converted to hex 0x37, and can be read. The holding register is at address 23 (hex 0x17), has a low value of 20, a high value of 150, an increment of 5, can be read, and can be written to singly only.

3. After plc.txt is configured to reflect the PLC, open and run mpack_creator.m. It creates five files:

   - read_mpacks.txt
   - write_mpacks.txt
   - a_check.bin
   - r_checkl.bin
   - r_checkh.bin

The first two are used to generate the Bloom filters while the last three need to be stored on the CompactFlash card for use in operating the software.

4. Next, run both bf_gen_k_10_m_16k_read.m and bf_gen_k_10_m_16k.m. They will generate coefficient files (.coe) that will be read by the Xilinx Core Generator to initialize the Bloom filters.

5. After the Bloom filter generator scripts have finished, navigate to the Modbus_Inspection_v2\ipcore_dir\directory. Copy the coefficient files (.coe) generated by Matlab into this directory, overwriting the old ones.

6. Next, open the Xilinx Core Generator program and open the project located in Modbus_Inspection_v2\ipcore_dir\. Click on “Project”, then “Regenerate all project IP (under current project settings)”.

7. After the IP cores have been regenerated, navigate back to Modbus_Inspection_v2\ipcore_dir\. Copy the generated bf_bram_*_*.ngc files into \pcores\modbus_inspector_core_v1_00_a\netlist , overwriting the files there. The Bloom filters are now ready to be loaded onto the FPGA!

C.3 Installation

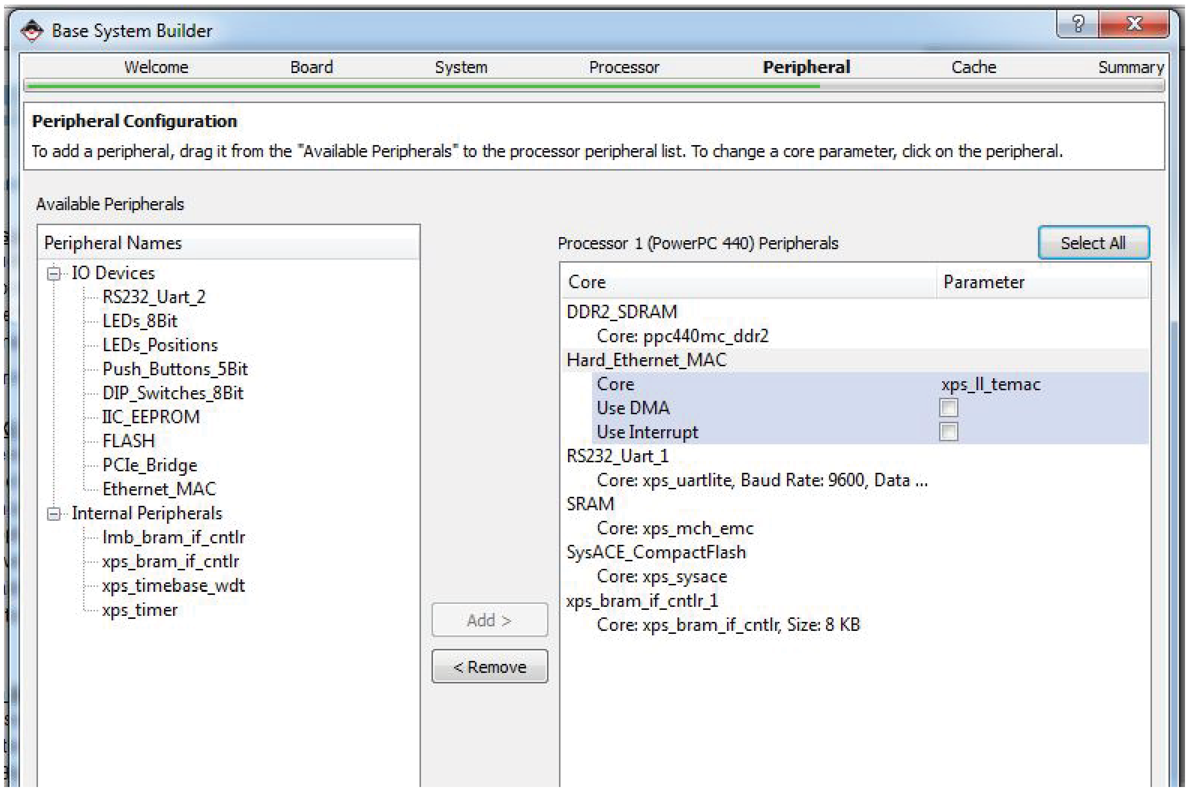
This section covers how to set up the hardware and install the software for the SIEVE system using the Xilinx ISE Design Suite, version 13.2.

1. First, open the Xilinx Platform Studio, select “Base System Builder wizard” and press “OK”.

2. Select a directory to store the project and set the radio button to “PLB System”. Click “OK”

3. Select “I would like to create a new design.” and click “Next”.

4. Select “I would like to create a system for the following development board”. From the dropdown for Board Vendor select “Xilinx”. From the dropdown for Board Name select “Virtex 5 ML507 Evaluation Platorm”. From the dropdown for Board Revision select ‘A’. Click “Next”.
5. Select “Single-Processor System”. Click “Next”.
6. From the dropdown for Processor Type select “PowerPC”. From the dropdowns for both Processor Clock Frequency and Bus Clock Frequency select “125.00”. From the dropdown for Debug Interface select “FPGA JTAG”. Click “Next”.
7. In the Peripheral Configuration window remove all peripherals except for DDR2_SDRAM, RS232_Uart_1, SRAM, SysACE_CompactFlash, and xps_bram_if_cntlr_1. Add Hard_Ethernet_MAC to the processor. The window should look like Figure C.1, below. Change the Baud Rate of the RS232_Uart_1 core from 9600 to 115200. Change the size of the xps_bram_if_cntlr_1 core from 8KB to 128 KB. Click “Next”.
8. Enable the check boxes for both the Instruction Cache and Data Cache. Enable only xps_bram_if_cntlr_1 under instruction cache and all three memories under Data Cache. Click “Next”.
9. Click “Finish”.
10. In the directory where the system was created, make a new folder called “pcores”. Copy the folders \adc_bank_core_v7_00_a, \custom_timer_core_v1_00_a, and \modbus_inspector_core_v1_00_a from the \pcores\folder on the CD into the newly created pcores folder.
11. In the Xilinx Platform Studio window, click Hardware, then “Rescan User Repositories”.



Peripheral Configuration

To add a peripheral, drag it from the "Available Peripherals" to the processor peripheral list. To change a core parameter, click on the peripheral.

Available Peripherals

Peripheral Names
- IO Devices
  - RS232_Uart_2
  - LEDs_8Bit
  - LEDs_Positions
  - Push_Buttons_5Bit
  - DIP_Switches_8Bit
  - IIC_EEPROM
  - FLASH
  - PCIe_Bridge
  - Ethernet_MAC
- Internal Peripherals
  - lmb_bram_if_cntlr
  - xps_bram_if_cntlr
  - xps_timebase_wdt
  - xps_timer

Processor 1 (PowerPC 440) Peripherals

Core	Parameter
DDR2_SDRAM	
Core: ppc440mc_ddr2	
Hard_Ethernet_MAC	
Core	xps_ll_temac
Use DMA	☐
Use Interrupt	☐
RS232_Uart_1	
Core: xps_uartlite, Baud Rate: 9600, Data ...	
SRAM	
Core: xps_mch_emc	
SysACE_CompactFlash	
Core: xps_sysace	
xps_bram_if_cntlr_1	
Core: xps_bram_if_cntlr, Size: 8 KB	

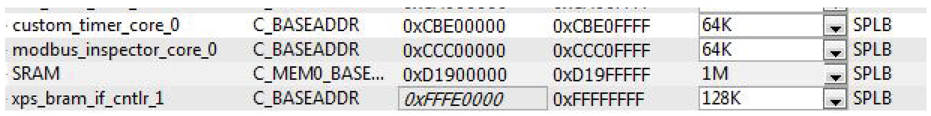
Figure C.1: Peripheral Configuration Window

12. On the left side of the Platform Studio window, click IP Catalog. Expand Project Local PCores, USER. Add the three IP cores to the project.

13. In the System Assembly View, on the Bus Interfaces tab expand each of the newly added cores. Set the SPLB dropdown to “plb_v46_0”.

14. Change to the Ports tab in the System Assembly View. Expand the adc_bank_core_0. Change the dropdown for each of the ports to “Make External”.

15. Change to the Addresses tab in the System Assembly View. Click the Generate Addresses button in the upper right corner. Change the uppermost byte of the Base Address of the SRAM from 8 to D, as shown below in Figure C.2.



custom_timer_core_0	C_BASEADDR	0xCBE00000	0xCBE0FFFF	64K	SPLB
modbus_inspector_core_0	C_BASEADDR	0xCCC00000	0xCCC0FFFF	64K	SPLB
SRAM	C_MEM0_BASE...	0xD1900000	0xD19FFFFF	1M	SPLB
xps_bram_if_cntlr_1	C_BASEADDR	0xFFFE0000	0xFFFFFFFF	128K	SPLB

Figure C.2: SRAM Address Setting

16. Copy the following entries into the system's .ucf file:

```
Net adc_bank_core_0_five_mhz_out_pin LOC=AH32;

Net adc_bank_core_0_adc_din_pin<9> LOC=H33;

Net adc_bank_core_0_conv_out_pin<9> LOC=F34;

Net adc_bank_core_0_adc_din_pin<8> LOC=H34;

Net adc_bank_core_0_conv_out_pin<8> LOC=G33;

Net adc_bank_core_0_adc_din_pin<7> LOC=G32;

Net adc_bank_core_0_conv_out_pin<7> LOC=H32;

Net adc_bank_core_0_adc_din_pin<6> LOC=J32;

Net adc_bank_core_0_conv_out_pin<6> LOC=J34;

Net adc_bank_core_0_adc_din_pin<5> LOC=L33;

Net adc_bank_core_0_conv_out_pin<5> LOC=M32;

Net adc_bank_core_0_adc_din_pin<4> LOC=P34;

Net adc_bank_core_0_conv_out_pin<4> LOC=N34;

Net adc_bank_core_0_adc_din_pin<3> LOC=AA34;

Net adc_bank_core_0_conv_out_pin<3> LOC=AD32;

Net adc_bank_core_0_adc_din_pin<2> LOC=Y34;

Net adc_bank_core_0_conv_out_pin<2> LOC=Y32;

Net adc_bank_core_0_adc_din_pin<1> LOC=W32;

Net adc_bank_core_0_conv_out_pin<1> LOC=AH34;

Net adc_bank_core_0_adc_din_pin<0> LOC=AE32;
Net adc_bank_core_0_conv_out_pin<0> LOC=AG32;
```

17. Click Project, then Export Hardware Design to SDK. This will take some time depending on the capabilities of the computer.
18. When the Xilinx SDK opens, click File, then New, then New Xilinx Board Support Package. Select “Standalone” for the Board Support OS and click “Finish”.
19. In the window that opens up, click the check box next to “xilfatfs”. A new xilfatfs configuration should appear in the window on the left. Select it, then change config_write’s value to TRUE. Click “OK”.
20. Copy the peripheral_tests_0 folder from the program folder on the CD into the SDK_Export directory of the directory the system is stored in.
21. Restart Xilinx SDK and the hardware and software installation is complete.

Bibliography

[AONR12] I. Ahmed, S. Obermeier, M. Naedele, and G. G. Richard. SCADA Systems: Challenges for Forensic Investigators. *Computer*, 45(12):44–51, 2012.

[Blo70] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, Jul 1970.

[Boy10] Stuart A. Boyer. *SCADA: Supervisory Control And Data Acquisition*. ISA, 4th edition, 2010.

[CCG+11] A. Carcano, A. Coletta, M. Guglielmi, M. Masera, I. N. Fovino, and A. Trombetta. A Multidimensional Critical State Analysis for Detecting Intrusions in SCADA Systems. *Industrial Informatics, IEEE Transactions on*, 7(2):179–186, 2011.

[CDF+07] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes. Using model-based intrusion detection for SCADA networks. In *Proceedings of the SCADA Security Scientific Symposium*, 2007.

[Cor98] Burr-Brown Corporation. *12-Bit High Speed Low Power Sampling ANALOG-TO-DIGITAL CONVERTER*, 1998. http://www.ti.com/lit/ds/symlink/ads7818.pdf.

[CW79] J. Lawrence Carter and Mark N. Wegman. Universal Classes of Hash Functions. *Journal of Computer and System Sciences*, 1979.

[CWY08] Ning Cai, Jidong Wang, and Xinghuo Yu. SCADA system security: Complexity, history and new developments. In *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, pages 569–574, 2008.

[Dev13] Analog Devices. *Single-Channel, 12-/16-Bit, Serial Input, 4 mA to 20 mA, Current Source DAC, HART Connectivity*, 2013. http://www.analog.com/static/imported-files/data_sheets/AD5410_5420.pdf.

[DKSL04] Sarang Dharmapurikar, Praveen Krishnamurthy, T. S. Sproull, and J. W. Lockwood. Deep Packet Inspection Using Parallel Bloom Filters. *Micro, IEEE*, 24(1):52–61, 2004.

[dnp12] *IEEE Standard for Electric Power Systems Communications–Distributed Network Protocol (DNP3)*, IEEE Std 1815-2012 (Revision of IEEE Std 1815-2010) edition, 2012.

[Dut07] Bruno Dutertre. Formal Modeling and Analysis of the Modbus Protocol. In Eric Goetz and Sujeet Shenoi, editors, *Critical Infrastructure Protection*, volume 253 of *IFIP International Federation for Information Processing*, pages 189–204. Springer US, 2007.

[EBPS09] Samuel East, Jonathan Butts, Mauricio Papa, and Sujeet Shenoi. A Taxonomy of Attacks on the DNP3 Protocol. In Charles Palmer and Sujeet Shenoi, editors, *Critical Infrastructure Protection III*, volume 311 of *IFIP Advances in Information and Communication Technology*, pages 67–81. Springer Berlin Heidelberg, 2009.

[FCAB00] Li Fan, Pei Cao, J Almeida, and A.Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *Networking, IEEE/ACM Transactions on*, 8(3):281–293, 2000.

[FCCM12] I. N. Fovino, A. Coletta, A. Carcano, and M. Masera. Critical State-Based Filtering System for Securing SCADA Network Protocols. *Industrial Electronics, IEEE Transactions on*, 59(10):3943–3950, 2012.

[FMC11] N. Falliere, L. Murchu, and E. Chen. W32.Stuxnet Dossier. Technical report, Symantec, 2011.

[fSC14] R Foundation for Statistical Computing. R, February 2014. http://www.r-project.org.

[GLLY10] Deke Guo, Yunhao Liu, XiangYang Li, and Panlong Yang. False Negative Problem of Counting Bloom Filter. *Knowledge and Data Engineering, IEEE Transactions on*, 22(5):651–664, 2010.

[HS05] Andrew Hildick-Smith. Security for Critical Infrastructure SCADA Systems. Technical report, Sans Institute, 2005.

[HSG13] Jeffrey L. Hieb, Jacob Schreiver, and James H. Graham. A security-hardened appliance for implementing authentication and access control in SCADA infrastructures with legacy field devices. *International Journal of Critical Infrastructure Protection*, 6(1):12–24, 3 2013.

[IEE08] *IEEE Standard for SCADA and Automation Systems*, IEEE Std C37.1-2007 (Revision of IEEE Std C37.1-1994) edition, 2008. http://standards.ieee.org/findstds/standard/C37.1-2007.html.

[ILW06] Vinay M. Igure, Sean A. Laughter, and Ronald D. Williams. Security issues in SCADA networks. *Computers & Security*, 25(7):498–506, 10 2006.

[LMV12] O. Linda, M. Manic, and T. Vollmer. Improving cyber-security of smart grid systems via anomaly detection and linguistic domain knowledge. In *Resilient Control Systems (ISRCS), 2012 5th International Symposium on*, pages 48–54, 2012.

[LRBS12] P. Lambruschini, M. Raggio, R. Bajpai, and A. Sharma. Efficient implementation of packet pre-filtering for scalable analysis of IP traffic on high-speed lines. In *Software, Telecommunications and Computer Networks (SoftCOM), 2012 20th International Conference on*, pages 1–5, 2012.

[Mar14] Adriano Monteiro Marques. Zenmap, February 2014. http://nmap.org/zenmap/.

[Mat14] Mathworks. Matlab v2013a, February 2014. http://www.mathworks.com.

[MJVD13] Thomas H. Morris, Bryan A. Jones, Raford B. Vaughn, and Yoginder S. Dandass. Deterministic Intrusion Detection Rules for MODBUS Protocols. In *System Sciences (HICSS), 2013 46th Hawaii Conference On*, pages 1773–1781, 2013.

[MKR12] C. Mukuntharaj, M Karthikeyeni, and V. Rajeshkumar. FPGA Based Network Intrusion Detection System Using Counting Bloom Filter. 3:733–739, July 2012.

[MLVD12] M. D. D. Moreira, R. P. Laufer, P. B. Velloso, and O. C. M. B. Duarte. Capacity and Robustness Tradeoffs in Bloom Filters for Distributed Applications. *Parallel and Distributed Systems, IEEE Transactions on*, 23(12):2219–2230, 2012.

[Mod06] Modbus Organization. *MODBUS Messaging on TCP/IP Implementation Guide*, October 2006.

[Mod12] Modbus Organization. *MODBUS Application Protocol Specification*, April 2012.

[MW13] Edward J. Markey and Henry A. Waxman. Electric Grid Vulnerability. Technical report, U.S. House of Representatives, 2013.

[NWD$^+$12] A. Nicholson, S. Webber, S. Dyer, T. Patel, and H. Janicke. SCADA security in the light of Cyber-Warfare. *Computers & Security*, 31(4):418–436, 6 2012.

[PK12] S. Parthasarathy and D. Kundur. Bloom filter based intrusion detection for smart grid SCADA. In *Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on*, pages 1–6, 2012.

[RFB97] M. V. Ramakrishna, E. Fu, and E. Bahcekapili. Efficient hardware hashing functions for high performance computers. *Computers, IEEE Transactions on*, 46(12):1378–1381, 1997.

[Roc12] Rockwell Automation. *1756 ControlLogix I/O Specifications*, 2012. http://literature.rockwellautomation.com/idc/groups/literature/documents/td/1756-td002_-en-e.pdf.

[SFS11] K. Stouffer, J. Falco, and K. Scarfone. Guide to Inustrial Control Systems Security. Technical report, NIST, 2011.

[SMPMLVLS11] Javier Sanchez-Monedero, Javier Povedano-Molina, Jose M. Lopez-Vega, and Juan M. Lopez-Soler. Bloom filter-based discovery protocol for DDS middleware. *Journal of Parallel and Distributed Computing*, 71(10):1305–1317, 10 2011.

[Sol] Murata Power Solutions. *4-20mA Current Loop Primer*. http://www.murata-ps.com/data/meters/dms-an20.pdf.

[SP05] Ioannis Sourdis and Dionisios Pnevmatikatos. *Fast, Large-scale String Match for a 10 Gbps FPGA-based NIDS*, pages 195–207. Springer US, 01/01 2005.

[Tcp14] Tcpdump. Tcpdump, February 2014. http://www.tcpdump.org/.

[TLM08] Chee-Wooi Ten, Chen-Ching Liu, and G. Manimaran. Vulnerability Assessment of Cybersecurity for SCADA Systems. *Power Systems, IEEE Transactions on*, 23(4):1836–1846, 2008.

[TM14] Inc. Triangle MicroWorks. Scada test harness, February 2014. http://www.trianglemicroworks.com/.

[Tur14] Aaron Turner. Tcpreplay, February 2014. http://tcpreplay.synfin.net/.

[VM08] J. Verba and M. Milvich. Idaho National Laboratory Supervisory Control and Data Acquisition Intrusion Detection System (SCADA IDS). In *Technologies for Homeland Security, 2008 IEEE Conference on*, pages 469–473, 2008.

[Wir14a] Wireshark. Libpcap file format, February 2014. http://wiki.wireshark.org/Development/LibpcapFileFormat.

[Wir14b] Wireshark. Wireshark network protocol analyzer, February 2014. http://www.wireshark.org.

[YML$^+$12] Y. Yang, K. McLaughlin, T. Littler, S. Sezer, Eul Gyu Im, Z. Q. Yao, B. Pranggono, and H. F. Wang. Man-in-the-middle attack test-bed investigating cyber-security vulnerabilities in Smart Grid SCADA systems. In *Sustainable Power Generation and Supply (SUPERGEN 2012), International Conference on*, pages 1–8, 2012.

[ZJS11] B. Zhu, A. Joseph, and S. Sastry. A Taxonomy of Cyber Attacks on SCADA Systems. In *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, pages 380–388, 2011.

[ZS10] B. Zhu and S. Sastry. SCADA-specific Intrusion Detection/Prevention Systems: A Survey and Taxonomy. In *In Proceedings of the First Workshop on Secure Control Systems*, 2010.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 27–03–2014	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Oct 2013–Mar 2014
4. TITLE AND SUBTITLE Integrity Verification for SCADA Devices Using Bloom Filters and Deep Packet Inspection	**5a. CONTRACT NUMBER**	
	5b. GRANT NUMBER	
	5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Doroski, Michael W., Captain, USAF	**5d. PROJECT NUMBER**	
	5e. TASK NUMBER	
	5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB, OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT-ENG-14-M-25	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Nick Carr DHS ICS-CERT 245 Murray Lane SW Bldg 410, Mail Stop 635 Washington, DC 20528 208-526-0900	**10. SPONSOR/MONITOR'S ACRONYM(S)**	
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)	

12. DISTRIBUTION / AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES
This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT
In the past, SCADA networks were made secure through undocumented, proprietary protocols and isolation from other networks. Today, modern information technology (IT) solutions have provided a means to enhance remote access through use of the Internet. Unfortunately, opening SCADA networks to the Internet has provided routes of attack. Cyber attacks on these networks are becoming more common and can inflict considerable damage to critical infrastructure systems. Furthermore, devices on these networks can be infected with malware that causes them to falsify their responses to operators, concealing alternate operation or hiding alarm conditions. Considering their applications, securing these networks translates to improved physical security in the real world.
Since modern IT solutions are impractical to deploy in the resource constrained SCADA networks, other solutions must be researched. This research evaluates an integrity verification system implemented on a Xilinx ML507 development board called the SIEVE system. The design incorporates Bloom filters and SCADA-specific intrusion detection techniques to speed identification of invalid commands and current sensing to investigate whether or not a device correctly carried out a given command.
Results show that the SIEVE system is able to inspect and correctly identify 100% of network traffic at a 200 command per second frequency. Correct identification of valid MODBUS/TCP traffic begins to fail at 350 commands per second, introducing false positives. Tests of the Bloom filters show that they reduce the time necessary to process and log invalid MODBUS/TCP commands by 4.5% to 2328.06% depending on the number of operations performed by the command.

15. SUBJECT TERMS
SCADA, Integrity Verification, Bloom Filters, FPGA, Deep Packet Inspection

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dr. Barry E. Mullins (ENG)
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U	UU	132	(937) 255-3636 x7979 barry.mullins@afit.edu